Tech Fundamentals

Project 2 Submission

Kaggle Competition: *Porto Seguro's Safe Driver Prediction*

Executive Summary

Overview:

Porto Seguro - One of the Brazil's largest auto and home insurance company sponsors this competition. The objective of this competition is to develop a machine-learning model that can predict the probability of claim initiation by a driver, based on certain features. Accurate prediction of claim initiation helps insurance companies in customizing their offerings for different drivers based on their respective risk profiles. Inaccuracies in claim prediction end up penalizing good drivers and rewarding bad ones.

Data:

Data consists of claim history of ~1.5MM customers with 57 anonymous attributes, broken into train (~0.6MM) and test (0.9MM) dataset. Attributes are a mix of binary (17), categorical (14) and continuous (26) types. Anonymous attributes ensure that the models are driven by data considerations only, without being influenced by any bias (intuition or actual market acumen). On the other side, anonymous attributes leave no space for the "analyst" within a data scientist to leverage his/her business acumen in solving this problem.

Summary of modeling:

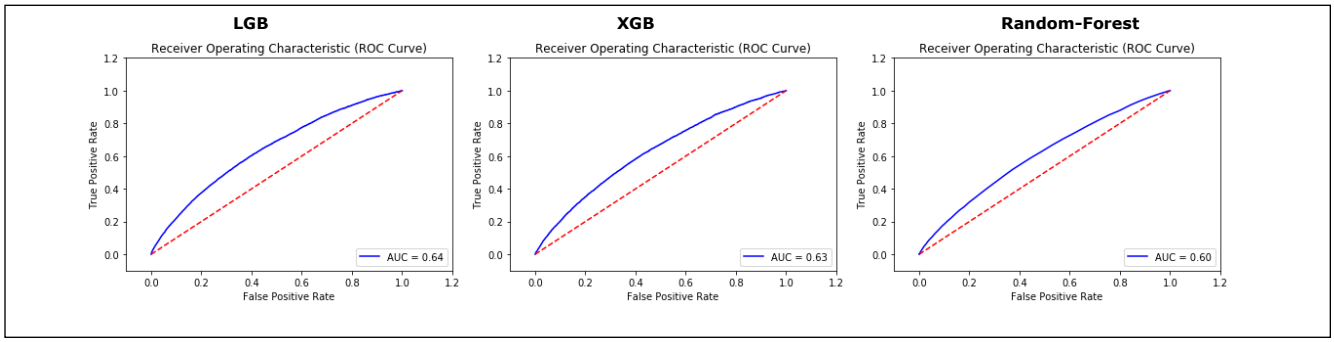
This is a highly imbalanced classification problem because only ~3.5% of the records are labelled positive. Therefore, I approached it with three models that are suitable for this scenario:

1. ***Random Forest with Synthetic Minority Over-sampling Technique (SMOTE)***
2. ***Extreme Gradient Boosting Model (XGBoost)***
3. ***Light Gradient Boosting Model (LGB)***

4.

The evaluation criteria for this competition is the Normalized Gini coefficient, which goes from 0 to 1, 0 is equivalent to a random classifier and 1 represents a perfect classifier.

Out of these three models, LGB (Gini Coef. on validation set = 0.284) performed the best, followed by XGboost (Gini Coef. = 0.259) and Random-Forest (Gini Coef. = 0.21). That is expected, as boosting models are the best performing models for classification problems. They have consistently outperformed other models in many Kaggle completions. Chart1, Chart2 and Chart3 below show the RoC curves for LGB, XGB and Random Forest models respectively. I have also included the area under the curve (AUC) in the chart area (bottom right corner).
Details of parameter tuning for each model is described in subsequent sections.



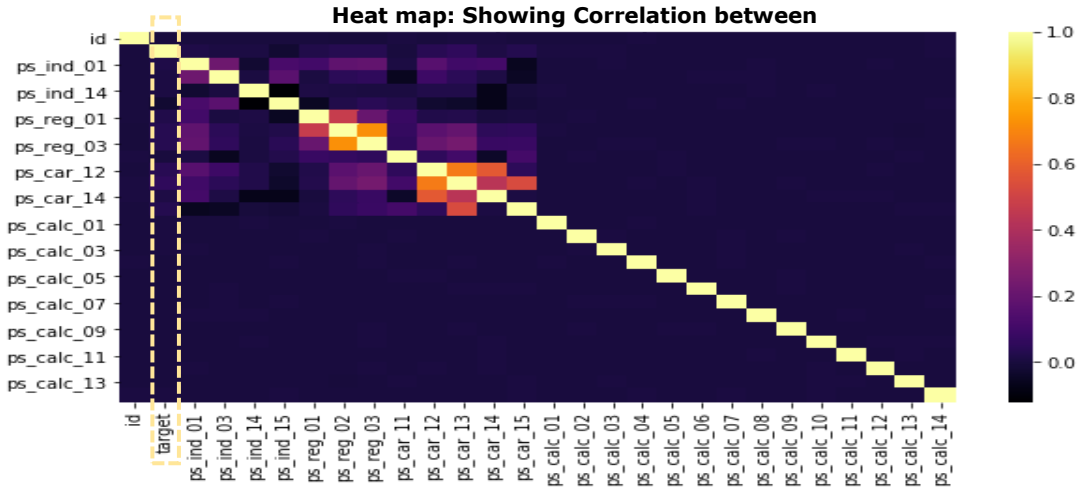
Analysis of Relevance of Independent Variables

First step to any kind of feature engineering is to get rid of the Null values in the dataset. For this analysis, I replaced all the null values with the median values of their respective attributes.

Since this dataset has a mixture of continuous, binary and categorical attributes, first I need to convert categorical attributes into numeric ones before I can use them in models. I used the one-hot encoding approach for this purpose. It converts a categorical attribute into (n-1) binary attributes where 'n' is the number of unique categories in that attribute. After one hot encoding, total number of attributes increased from 57 to 204. Throughout this process, I ensured that I applied the exact same transformations to both training and testing datasets, to ensure consistency in the schema of the two datasets.

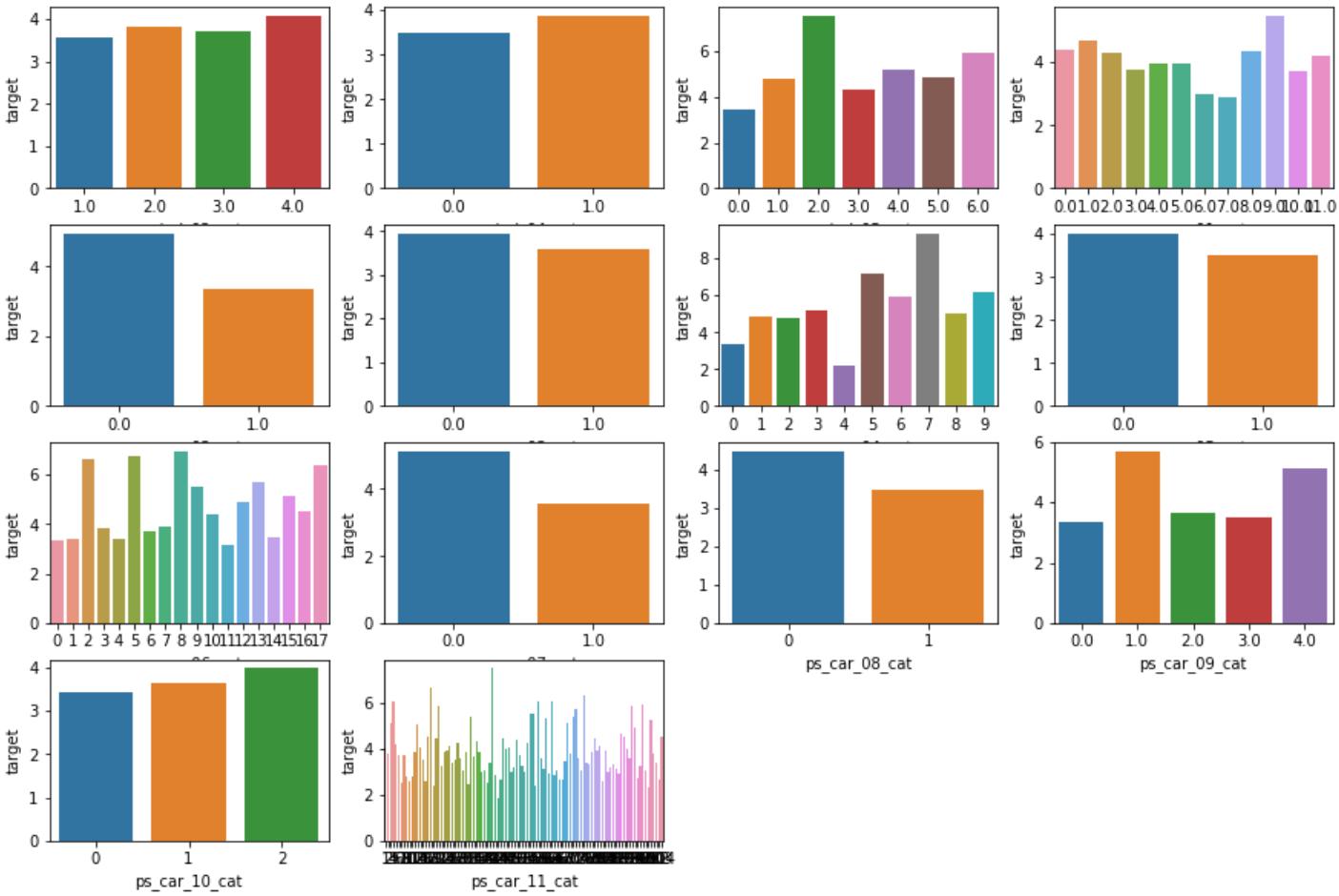
Since the attributes in this dataset are anonymous, there is very little space for any feature engineering based on intuition or logic; therefore, first I tried to utilize some of the mathematical approaches to reduce the dimensionality of this problem. I tried Principle Component Analysis (PCA), which reduces the dimensionality of the dataset by mapping the original data space to a transformed space with significantly lesser dimensions while ensuring a near 100% coverage of variance in the original dataset. However, I found out that PCA is not an effective method for dimensionality reduction in a categorical data. In fact, there is no foolproof method available to reduce the dimensionality of categorical attributes.

Then I tried to explore if I could find some correlation between target attribute and any of the categorical ones. In my explorative data analysis for project 1, I could not find significant correlation between any of the continuous attributes and the target binary classes, as is clear from the heat map below. Please see region highlighted with dotted yellow line:



Now, I tried to see if there is any category in any of the categorical attributes, which explains significantly higher percentage of true positives than others do. Following charts show the contribution of true positive labels to the total count of individual categories in each categorical attribute. Unfortunately, there is no category, which has particularly high correlation with the true positive class in target attribute. Almost all the categories have a contribution of true positive class in range 2.5% to 8% of total observations. If there was a category with high correlation, we could have used that information while choosing the right model for this problem and model tuning thereafter. Please refer the set of bar charts below.

Percentage contribution of true positives corresponding to each category of categorical attributes:



Therefore, in absence of any particular information, I decided to go ahead with all the available attributes that I got after one hot encoding. Which means that I have 204 independent attributes in the dataset. Although 204 is a large dimensionality, python has the ability to handle such large datasets with quite an ease. Especially, since I used Kaggle's python docker platform to develop my code, it was very easy to run models on dataset with 204 attributes.

Analysis of Models used:

First step to any modeling process is to break down the input training dataset (0.6 MM records) into train (0.4MM records) and validation (0.2MM records) sets. Train set is used to train the model and the validation set to test the performance of the learned model. The best performing model on validation set were then tested on the actual testing dataset (~0.9 million records in this case). There is an in built method in sklearn.model_selection library in python called "train_test_split" which splits the train dataset into internal training and validation sets.

Evaluation criteria for this completion is the normalized Gini coefficient. It is a derivative of the RoC AUC curve.

$$Normalized\ Gini\ Coefficient = 2\ X\ (ROC\ AUC) - 1$$

Therefore, maximizing the area under the curve (AUC) in RoC curve ensures maximization of the Gini coefficient. Normalized Gini coefficient ranges from 0 to 1, where 1 represents a perfect model which can predict the outcome with 100% accuracy and zero represents a random prediction model.

I used normalized Gini coefficient on validation set as an evaluation criteria for parameter tuning. The best performing models of each type were tested on actual testing set to get the final the Kaggle scores.

Although, there are many parameters in each model which can be tuned to improve the performance of the models, I chose to tune a selected few from them for this project.

Model tuning

1. Random Forest with Synthetic Minority Over-sampling Technique (SMOTE)

Since it is a highly imbalanced dataset, I used SMOTE technique, which systematically oversamples the minority class in an imbalanced data to make the ratio between the two classes close to one. I used the 'oversampling' method from "imblearn" library of python for this purpose. This technique artificially inflated the train dataset from 0.4 MM records to ~ 0.8MM records with ratio between the two classes close to one.

Random-Forest approach is an ensemble approach, which leverages learnings from multiple decision tree models also known as "base models" for random forest.

Key Parameters tuned:

1. ***Number of iterations*** **:** # of iterations to be used to optimize the individual base models
2. ***Sampling methodology*** **:** (Bootstrap or Sampling without replacement)

Table below shows the Gini coefficients corresponding to different choices of model parameters:

	# Iterations	Sampling Methodology	Gini Coefficient on validation set	Kaggle Score	
1	100	Bootstrap	0.162		
2	100	Sampling Without Replacement	0.155		
3	400	Sampling Without Replacement	0.201	0.18	⬅ Best Model
4	400	Bootstrap	0.21		

2. Extreme Gradient Boosting (XGBoost)

This model has consistently dominated Kaggle competitions involving classification problem on structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. This is an ensemble approach, which leverages the learnings from many weak decision tree models (also known as base models) to come up with weighted average prediction score where weights are calculated based on the quality of prediction from individual base models. The average model turns out to be much better than the constituent base models.

Key parameters tuned:

1. ***Learning rate*** **(η) :** This is the learning rate of the model from one iteration to another
2. ***Max Depth*** **:** Maximum depth of the decision-tree base model. This can also be interpreted as the dimensionality of the constituent decision tree models
3. ***Number of iterations*** **:** # of iterations to be used to optimize individual base models

Table below shows the Gini coefficients corresponding to different choices of model parameters:

	Learning Rate	Max Depth	# Iterations	Gini Coefficient on validation set	Kaggle Score	
1	0.01	20	100	0.257	-	
2	0.01	10	100	0.256	-	
3	0.01	10	500	0.259	0.234	← Best Model
4	0.1	10	500	0.17	-	

3. Light Gradient Boosting (LGB)

LGB is the lighter and more efficient version of the XGB model. From what I have understood, it is kind of an approximation of the true XGB model, which is significantly faster than the later but still gives comparable performance. Interestingly, for this analysis, LGB tends to outperform XGB.

Key parameters tuned:

1. ***Learning rate (η)*****:** This is the learning rate of the model from one ieteration to another
2. ***Max Depth:*** Maximum depth of the decision-tree base model. This can also be interpreted as the dimensionality of the constituent decision tree models
3. ***Number of iterations***: # of iterations to be used to optimize individual base models

Table below shows the Gini coefficients corresponding to different choices of model parameters:

	Learning Rate	Max Depth	# Iterations	Gini Coefficient on validation set	Kaggle Score	
1	0.01	10	100	0.261		
2	0.01	10	500	0.284	0.267	← Best Model
3	0.01	20	500	0.285		
4	0.1	5	500	0.281		
5	0.1	5	500	0.277		

Conclusion:

Overall, it was a great learning experience, which helped me get a grasp on some of the nuances of a data science project. The more time I spend on Python programming, the more I appreciate the immense capabilities of this language for data science. Python has made it so easy to use complex and powerful machine learning algorithms, especially for those with limited exposure to the underlying mathematical concepts. Although I still feel that one needs to have a basic understanding of the underlying mathematical concepts in order to utilize the full potential of these algorithms.

Another key takeaway is, that the more complex, and computationally expensive models need not always produce better results than a simpler model, as is clear from the fact that a simpler LGB model outperformed XGB model.