Machine Learning Engineer Nanodegree

Capstone Project

ANSHUL PRAKASH

Contents

1. Definition

Project Overview

With predictive analytics and machine learning being the buzzword today, each organization wants to know their customer better to provide them the best products and services at the right price. This understanding of customers involves not only getting insights from the past activities but also to predict the future activities as accurately as possible. This is of utmost important when it comes to pricing auto insurance as predicting probable future claims can be extremely helpful in pricing the products accordingly [1]. However, this power of pricing using the predictive power of machine learning algorithm comes with a caveat as an error in predicting can be really damaging specially when the algorithm predicts that a customer will make a claim and that does not happen. This results in an overpriced insurance bill for a customer who may have a safe driving record [3].

So, in this project I have tried to build a model to predict the probability that a driver will initiate an auto insurance claim next year. This is similar to the aim of "Porto Seguro's safe driver prediction" competition on Kaggle [4].

Domain Background

Auto insurance protects a user against monetary loss if they have an accident. It is a contract between the user and the insurance company. The user agrees to pay the premium and the insurance company agrees to pay losses as defined in the policy. Predictive analytics is used in appraising and controlling risk in underwriting, pricing, rating, claims, marketing and reserving in insurance sector [2].

In underwriting, predictive analytics enable better risk assessment and classification which leads to better pricing. Pricing analytics can be the "tipping point' for consumers in choosing insurers and products. They also lead to better profitability for insurers as they target desirable customer segments and support real time, dynamic pricing [2].

The use of predictive analytics has become more common in the insurance industry in recent years. Predictive analytics for insurance entails the use of special technology to sift through and analyze historical data and consumer trends in effort to project future behavior [1]. By studying the behavioral tendencies of varying demographics under differing sets of environmental circumstances, companies can learn what products those people might be inclined to buy, how best to reach them and most importantly what should be the price of the product so that it is a win-win situation for both the user and the company.

Porto Seguro is one of Brazil's largest auto and homeowner insurance companies. Porto Seguro has used machine learning for the past 20 years to offer the right product to right customer.

Problem Statement

Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones. The challenge is to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year.

A more accurate prediction will allow Porto Seguro to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers.

Datasets and Inputs

The dataset [4] provided on Kaggle consists of both training and testing data. In training data each row corresponds to a policy holder, and the target columns signifies that a claim was filed.

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.

Metrics

Submissions for this challenge on Kaggle are evaluated using the Normalized Gini Coefficient.

During scoring, observations are sorted from the largest to the smallest predictions. Predictions are only used for ordering observations; therefore, the relative magnitude of the predictions are not used during scoring. The scoring algorithm then compares the cumulative proportion of positive class observations to a theoretical uniform proportion.

The Gini Coefficient ranges from approximately 0 for random guessing, to approximately 0.5 for a perfect score. The theoretical maximum for the discrete calculation is (1 - frac_pos) / 2.

The Normalized Gini Coefficient adjusts the score by the theoretical maximum so that the maximum score is 1.

Gini Coefficient has been historically used in economics as a measure of statistical dispersion intended to represent the income or wealth distribution of a nation's residents, and is the most commonly used measure of inequality [10]. However, it is now being used widely with prediction models to measure the quality of model in an objective and quantifiable way.

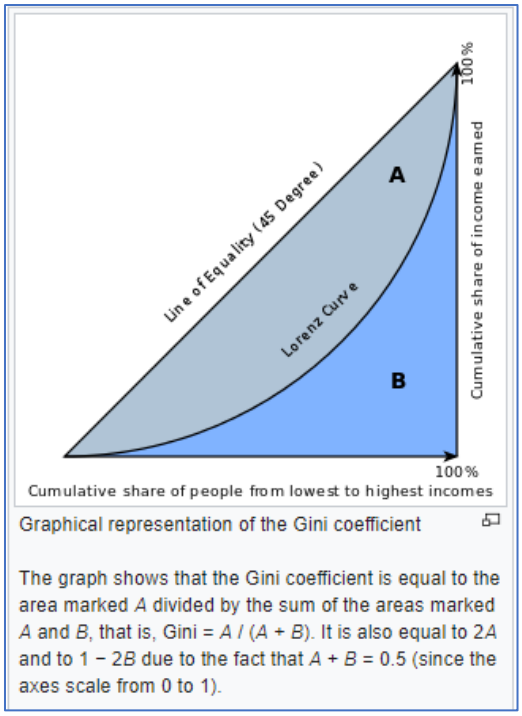


Figure 2: Graphical representation of Gini Coefficient [10]

The metric used in this competition is appropriate given the insurance domain of the problem. For calculating the gini coefficient the probabilities are sorted first and in then the cumulative proportion is compared to a uniform proportion. In this way it measures the ability of the model to rank order risk and this is really important for pricing insurance products for customers as discussed in the project overview.

Strategy

Such predictive analytics problem is best solved if approached with in a planned way with a strategy. The usual strategy employed is to first explore data, then to pre-process it, then various machine learning models are applied to get results and based on evaluations using the metric employed refinements are made to the model. Output from each stage is an input to the next stage and after data exploration the process is iterative to get the best results.

I also planned to use a strategy like this. I planned to use Python packages like numpy and pandas to analyze the given data sets and then to use packages like pyplot, seaborn and plot.ly to visualize the data to get some useful insights like correlation between features, amount of missing data etc. Based on the exploration done, data will be pre-processed to clean the data and to transform it into forms which is required by the models to be used for predicting. Results of exploration phase also help in deciding the models to be used for example if the data is of high dimensions then we can rule out use of certain models as they need lot of data and time. Then the predictions made using the model are evaluated using the metrics and then refinements are made to models iteratively to improve the results.

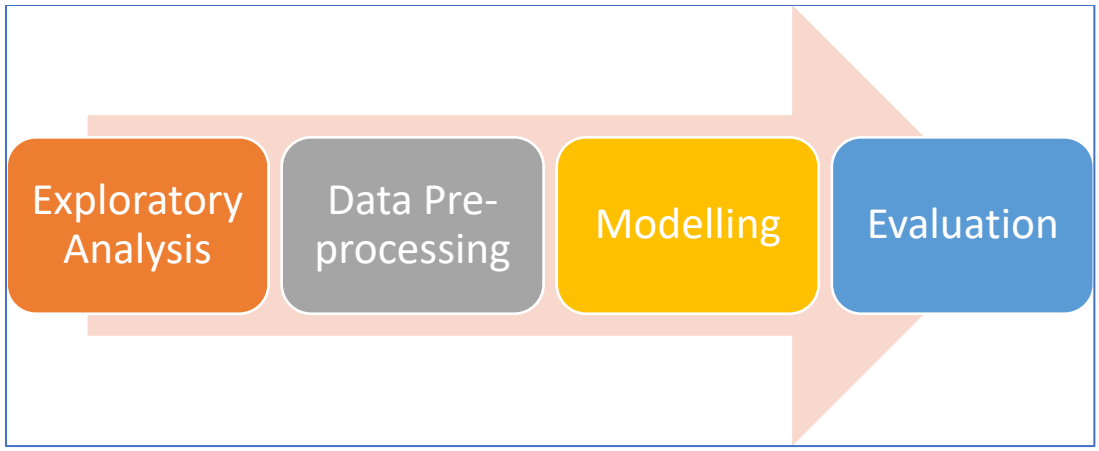


Figure 1: Strategy

2. Analysis

Data Exploration

Train and test data was provided in form of csv files. The first 5 rows of training data are as shown below:

	id	target	ps_ind_01	ps_ind_02_cat	ps_ind_03	ps_ind_04_cat	ps_ind_05_cat	ps_ind_06_bin	ps_ind_07_bin	ps_ind_08_bin
0	7	0	2	2	5	1	0	0	1	0
1	9	0	1	1	7	0	0	0	0	1
2	13	0	5	4	9	1	0	0	0	1
3	16	0	0	1	2	0	0	1	0	0
4	17	0	0	2	0	1	0	1	0	0

5 rows × 59 columns

Figure 2: First 5 rows of training data

Rows have been truncated for visual purpose as there are 59 columns. Training and testing data has unique id for each customer and all the related features are present for a customer in a row. Only the training data has a column ‘target’ with 0 indicating that a claim was not filed by the customer and ‘1’ indicating that a claim was filed next year.

Some important characteristics of the data are as follows:

1. The train dataset contains 595212 rows and 59 columns
2. The test dataset contains 82816 rows and 58 columns
3. Null values have been replaced by -1
4. Column names just give an idea of type of variable and not much information is provided about what the feature means.

Data type check

There are only two data types – float and integer. In training data there are 49 columns of type integer whereas 10 columns of type float. However categorical data has data type as integer. So, making use of the fact that categorical data features are appended with “cat” postfix I identified categorical data separately.

Meta Data

Data provided by Porto Seguro has suffixes with abbreviations such as "bin", "cat" and "reg", where:

- bin indicates binary features
- cat indicates categorical features
- rest are either continuous or ordinal features.

Metadata for data was maintained under the following labels:

- role: input, ID, target
- level: nominal, interval, ordinal, binary
- dtype: int, float, str

Grouping training data as per the labels mentioned above provides the following summary for data:

	role	level	count
0	id	nominal	1
1	input	binary	17
2	input	interval	10
3	input	nominal	14
4	input	ordinal	16
5	target	binary	1

Figure 3: Meta data

Exploratory Visualization

I made visualizations using a tutorial provided for plot.ly [5].

Missing Values

Since, all the missing values have been replaced by -1, using regular python commands to find if there any missing values was misleading.

```
# Checking for null values
train.isnull().any().any()
```

```
False
```

Figure 4: Check for null values

So, all the -1 values were replaced by NaN and the below visualization was created using plot.ly library to assess the amount of data that was missing:

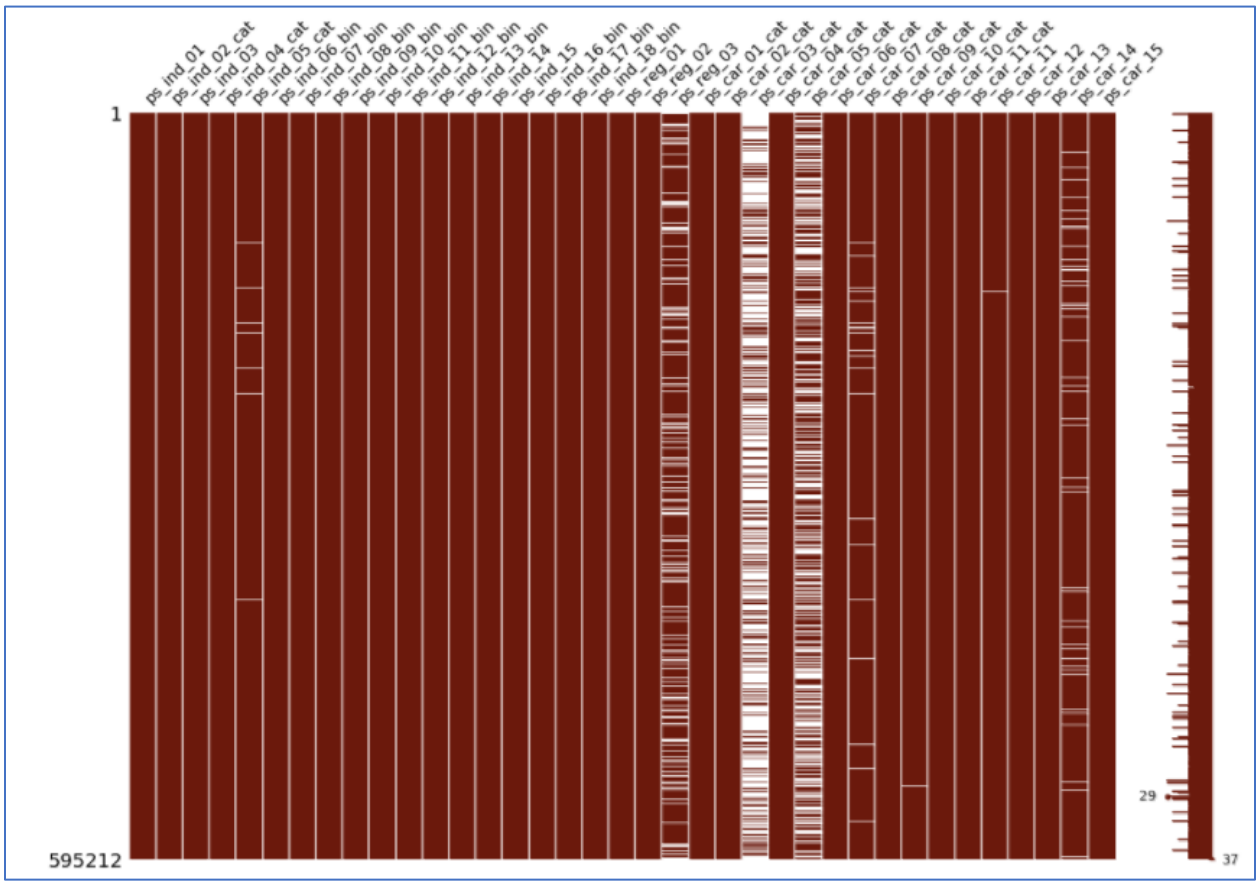


Figure 5: Visualization of missing data

In this visualization missing data is shown by white bands superimposed on dark red bands that show the values that are not missing or null. This visualization is really helpful in order to get a visual estimate of amount of data that is missing and to clearly find out which features have the most missing values. However, this visualization excludes certain null features as it can only fit in approximately 40 odd features. So, from the visualization we can see that ps_reg_03, ps_car_03_cat and ps_car_05_cat have the most missing values. The list of features with missing values is as below:

Columns	Number of NaN
ps_ind_02_cat:	216
ps_ind_04_cat:	83
ps_ind_05_cat:	5809
ps_reg_03:	107772
ps_car_01_cat:	107
ps_car_02_cat:	5
ps_car_03_cat:	411231
ps_car_05_cat:	266551
ps_car_07_cat:	11489
ps_car_09_cat:	569
ps_car_11:	5
ps_car_12:	1
ps_car_14:	42620

Figure 6: Number of NaNs

ps_car_03_cat, **ps_car_05_cat**, **ps_reg_03**, **ps_car_14** and **ps_car_07_cat** have missing values for more than 10,000 rows in training data. Many null values will cause error in training of models as they will lead to wrong predictions or models not working at all.

Target Exploration

The distribution of target variable in the training data is as shown below:

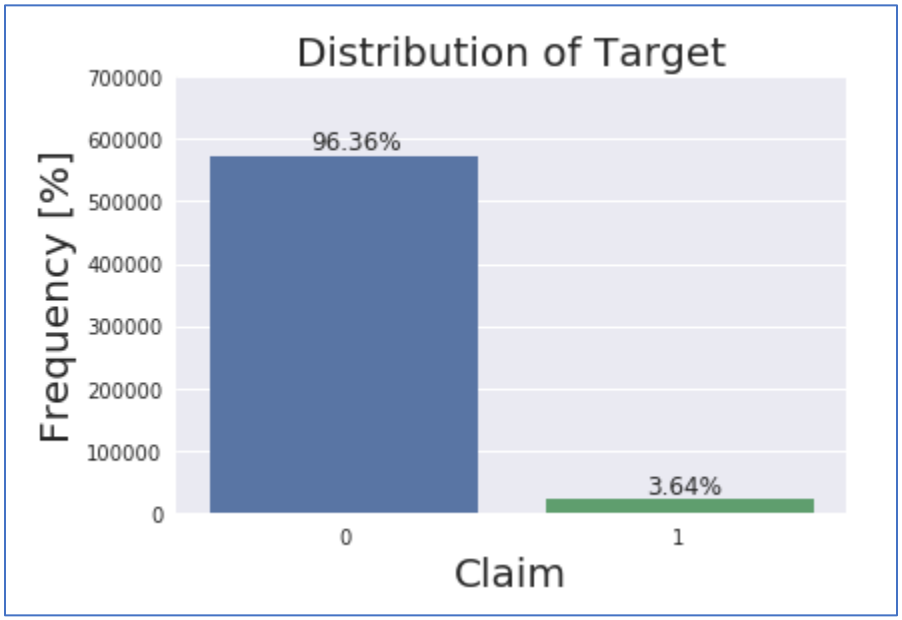


Figure 7: Target Distribution

Value for target is 1 for 3.64% of the records in training data. So, if we use a naive classifier that simply classifies target as 0 for all the rows then the prediction accuracy will be very high. Also, if we train a model using such imbalanced data then it will have high accuracy as the model will be biased to label data as 1 regardless of the data it is asked to predict. We will have to use a strategy to overcome this problem.

Correlation Plots

Correlation plots are useful to check if there is a high correlation between variables or not. This will be useful in deciding if PCA would be helpful or not.

For features with float data type, the correlation plot obtained using plot.ly library was as below:

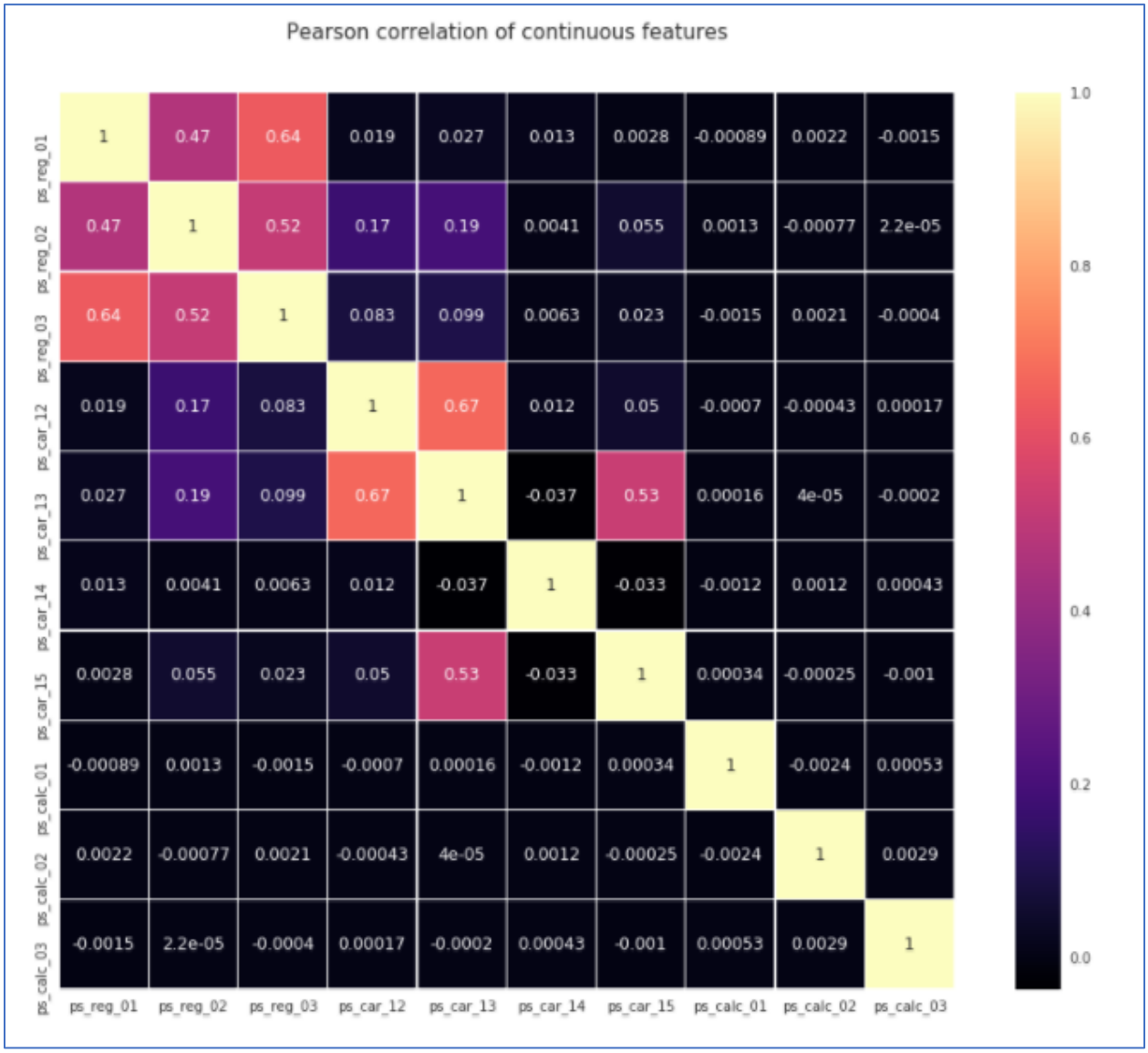


Figure 8: Correlation plot for continuous features

Majority of the features display zero or no correlation to one another. Only paired features such as (ps_reg_01, ps_reg_03), (ps_reg_02, ps_reg_03), (ps_car_12, ps_car_13), (ps_car_13,

ps_car_15) have high correlation. So, it might not make any difference if we do PCA on the training data as the number of correlated variables is low.

Similarly, for features with integer datatype the correlation plot obtained is as below:

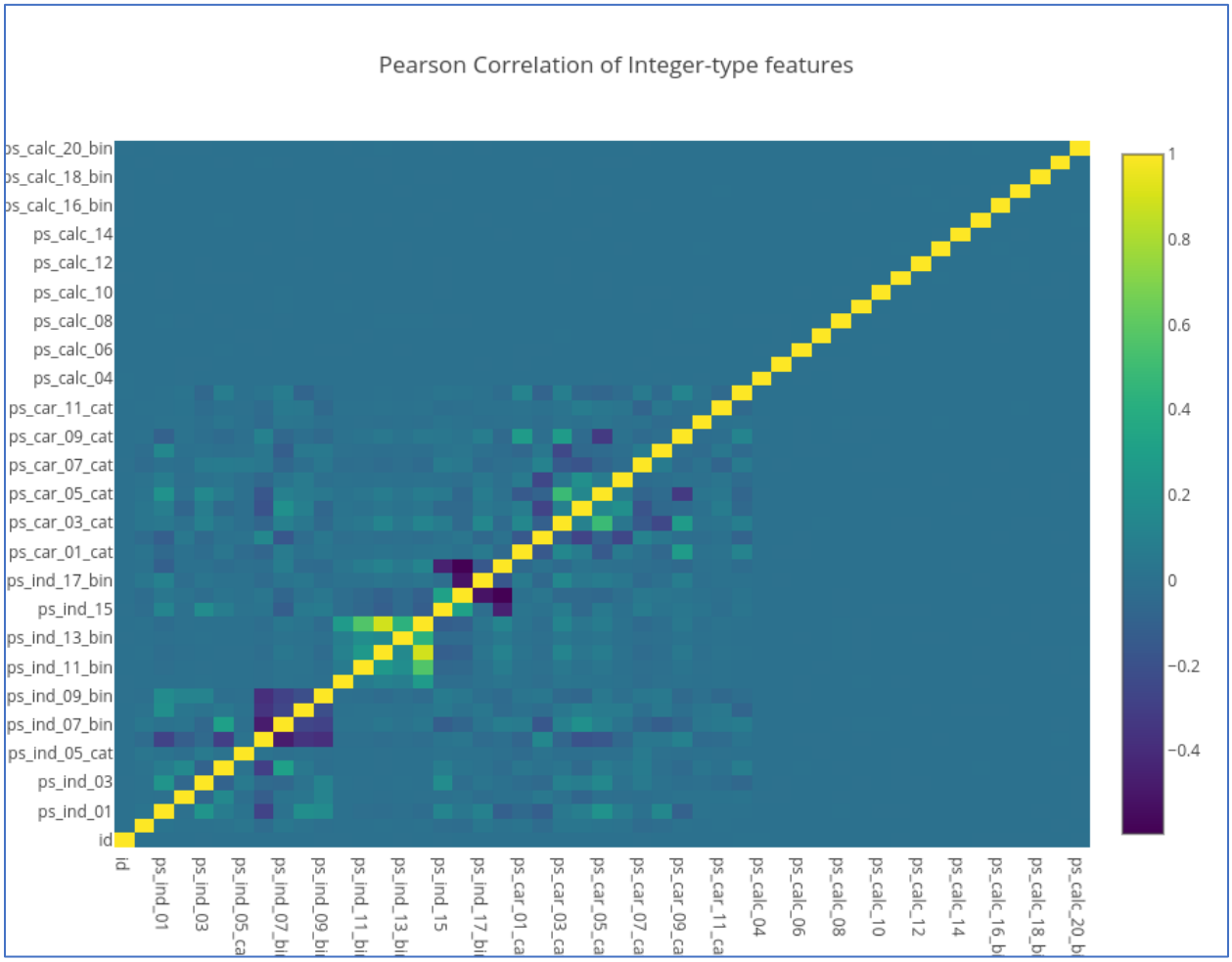


Figure 9: Correlation plot / Heat map for integer like features

The heat map shown above uses plot.ly as described by <>. x and y axes take in the column names while the correlation value is provided by the z-axis. It is quite evident that a huge number of columns with integer datatype are also not linearly correlated.

An important type to note here is that features starting with “ps_calc_” are not related to target at all. These features can be dropped while refining the models.

Binary Features Inspection

A plot of columns with postfix 'bin' was created to inspect the count of 1s and 0s. The plot is as shown below:

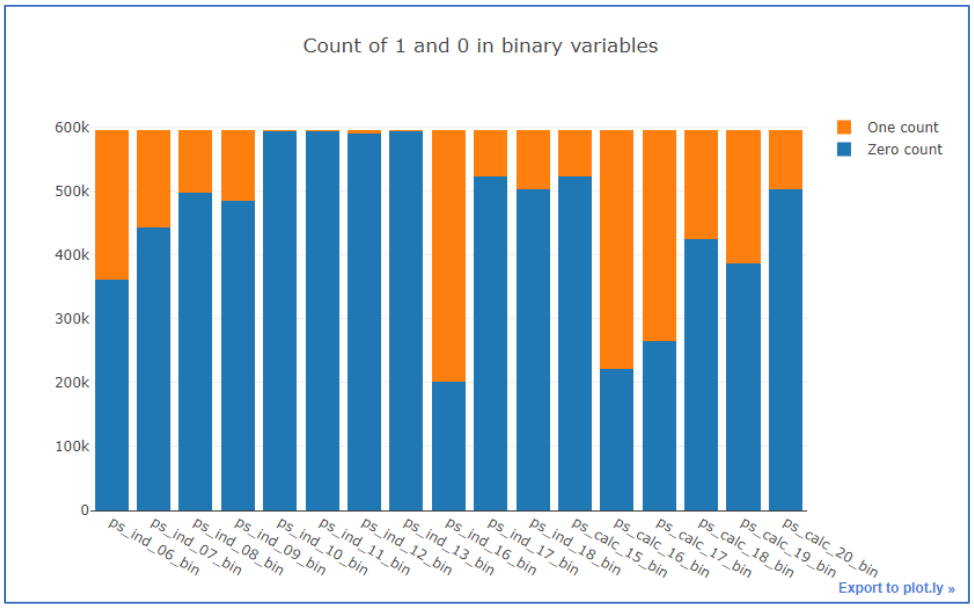


Figure 10: Binary features bar plot

From this plot it can be observed that **ps_ind_10_bin, ps_ind_11_bin, ps_ind_12_bin, ps_ind_13_bin** are completely dominated by zeros.

Categorical and Ordinal Feature Inspection

Cardinality refers to the number of different values in a variable. As we will create dummy variables from the categorical variables later on, we need to check whether there are variables with many distinct values.

```
Variable ps_ind_02_cat has 5 distinct values
Variable ps_ind_04_cat has 3 distinct values
Variable ps_ind_05_cat has 8 distinct values
Variable ps_car_01_cat has 13 distinct values
Variable ps_car_02_cat has 3 distinct values
Variable ps_car_03_cat has 3 distinct values
Variable ps_car_04_cat has 10 distinct values
Variable ps_car_05_cat has 3 distinct values
Variable ps_car_06_cat has 18 distinct values
Variable ps_car_07_cat has 3 distinct values
Variable ps_car_08_cat has 2 distinct values
Variable ps_car_09_cat has 6 distinct values
Variable ps_car_10_cat has 3 distinct values
Variable ps_car_11_cat has 104 distinct values
```

Figure 11: Cardinality for categorical features

Only ps_car_11_cat has many distinct values, although it is still reasonable.

Algorithms and Techniques

Classification vs Regression

This is clearly a classification problem as the participants must predict whether a customer will make a claim or not. But instead of clearly labelling the prediction as 0 or 1 submitting the probabilities would be enough as the predictions submitted will be used to calculate normalized gini coefficient for evaluation.

The training data is highly imbalanced, and a good benchmark might have been a naïve model that always predicts 0 as only 3.64% of the training records have a label of 1. But as gini coefficient is being used such a naïve model is of no use. Predicting using such a model gives a normalized gini coefficient of around 0.

Models

First, I planned to create naïve models that predict all zeroes, ones and random number between 0 and 1. Extreme Gradient boosting models are the ones which usually do well in competitions on Kaggle. But I decided to try all the classification models learned in the nanodegree program based on their advantages and disadvantages.

1. **Logistic Regression**: It is a classification tool which falls under the generalized linear modeling framework. It provides probabilities associated with class membership which are calculated using a parametric form assumed for the model. Maximum likelihood estimation is used to solve for the parameters that best fit the data.

   Logistic regression models are fast and have a very good explanatory value. But they are very biased, assuming that each predictor affects the log-odd of the outcome linearly and additively. Also, they don't handle missing values well and have trouble with large categorical variables.

2. **KNN**: These models work using the basic idea that "If it walks like a duck, quacks like a duck, then it is probably a duck". All observations in these models correspond to points in an Euclidean space of dimensions equal to the number of predictors. Classification is done by comparing the feature vectors of the different points.

   KNN models work really well for a small number of predictors. But they are not preferred for data with more than 30 predictors as due to curse of dimensionality data required is very high. Also, they require a lot of memory and they are computationally inefficient.

3. **Decision Trees**: These models make use of a "greedy" recursive partitioning algorithm to generate trees where each internal node tests attributes and branches correspond to attribute values. Each leaf node assigns a classification based on most commonly occurring class of observations in the leaf node. Starting from the root node the algorithm considers all partitions using the predictors and chooses the partition which reduces the node impurity measured by entropy or Gini-index the most. In this way attributes which are most informative are used first to get the best splits.

Decision trees are highly interpretable, robust to redundant and correlated variables and they handle missing values readily. But they easily overfit data and often do not predict well. The tree structure is very sensitive to training data.

4. **Random Forest**: These models overcome the problems associated with decision trees at the cost of reduced interpretability. Many trees are created using a random set of predictors each time and the resulting predictions are averaged. Selecting random set of predictors decorrelates the trees formed and this along with averaging reduces the variance of the model. These models come under Ensemble learning methods where multiple learners are trained to solve the same problem.

   Random forest models with different flavors are the model of choice for competitions on Kaggle as they work really well with high dimensional predictors and large training sets. They are easy to tune and provide importance of predictors which can be further used to get the best model. However, large number of trees may make real time prediction slow.

5. **Boosted Trees**: Boosting is a machine learning ensemble meta-algorithm in which a number of weak learners are used to create a strong one [12]. Boosted tree models build trees one at a time and these models are very useful as with each new tree model improvises on errors made in previous trees. The base learner takes all the distributions and assigns equal weight to each observation. If there is any prediction error caused by first base learning algorithm, then we pay high attention to observations having prediction error. These steps are repeated until high accuracy is achieved or when there are no considerable improvements [13]. However, as the trees are built sequentially building boosted trees takes a lot of time [7].

So, with predictors greater than 50 I decided not to use KNN. Also, since decision trees have low accuracy and are prone to overfitting I decided not to use them.

My strategy was to first use logistic regression only if the categorical variables were not large enough as all advantages of logistic regression model are lost if the categorical variables have a lot of levels.

Random Forest model looks the best for this type of data and I planned to use it initially in order to get a start and to know variable importance. Finally, I planned to try my hand at using XGBoost as refinement to random forest models.

Benchmark

The maximum possible score using a normalized gini coefficient is 0.50. My aim was to achieve a score of about 0.28 – 0.29 as the top score was greater than that. Finally, the winning score was 0.29698.

3. Methodology

Data Preprocessing

Data preprocessing is an important step as there are missing values in data, data is highly imbalanced, and some models have specific requirements for training data. Data pre-processing performed was as follows:

Missing Values

As evident through the exploratory analysis there are 13 columns with varying levels of missing data and most of them are categorical in nature.
ps_car_03_cat, ps_car_05_cat, ps_reg_03, ps_car_14 and ps_car_07_cat have missing values for more than 10,000 rows. I implemented two strategies:

1. Missing values were replaced by median values of rest of the data as median value are the most robust representation of data.
2. Columns with large amount of missing data were dropped.

Second option was used to improve the models.

Imbalanced Data

There are many ways to handle imbalanced data and resampling techniques are the most commonly used [6]. In resampling techniques classes are balanced in training data before providing as input to the machine learning model. We can resample in two ways:

1. **Under Sampling**: In this we eliminate records of majority class randomly to balance the two classes. It can help improve run time by reducing the number of training data samples when the training data set is huge. But at the same time, it can lead to information loss as it discards some of the records. Also, the sample chosen by random under sampling may be a biased sample.
2. **Over Sampling**: In this the number of instances of minority class are increased by replicating the records of minority class. This leads to no information loss but at the same time increases chances of overfitting.

Since there are 595212 rows in training data, I decided to do under sampling in order to avoid overfitting of the data.

Handling categorical data

Dummy variables were created for categorical columns. 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 were needed to deal with that. Shape of training data and testing data after creating dummy variables were as shown below:

```
(216940, 210)
(892816, 208)
```

Dropping Columns

As was evident in exploratory analysis that columns such as those starting with 'ps_calc' are not related to target at all and are a good candidate for dropping in order to improve the models.

Implementation

Predicting all zeroes/ ones/ random

Numpy's ones and zeroes function were used to create data frames with all predictions as zeroes and ones respectively for ids of test data. Python's random package was used to create data frames with random predictions. The scores obtained for all 3 were below zero and this was expected as normalized gini coefficient is being used for evaluating predictions of test data.

Random Forest

Version 1

First, I did the following pre-processing:

1. Replaced missing values of a columns with median value for a column.
2. Created dummy variables for categorical data
3. Divided the training data using `train_test_split` function of sklearn

Model parameters were:

- number of estimators: 200
- out of bagging set to True
- N_jobs: Use all the available cores= -1
- min_sample_leaf: minimum number of samples required to be at a leaf node was set to 100

Score obtained using this model: 0.26504

Confusion matrix for training data:

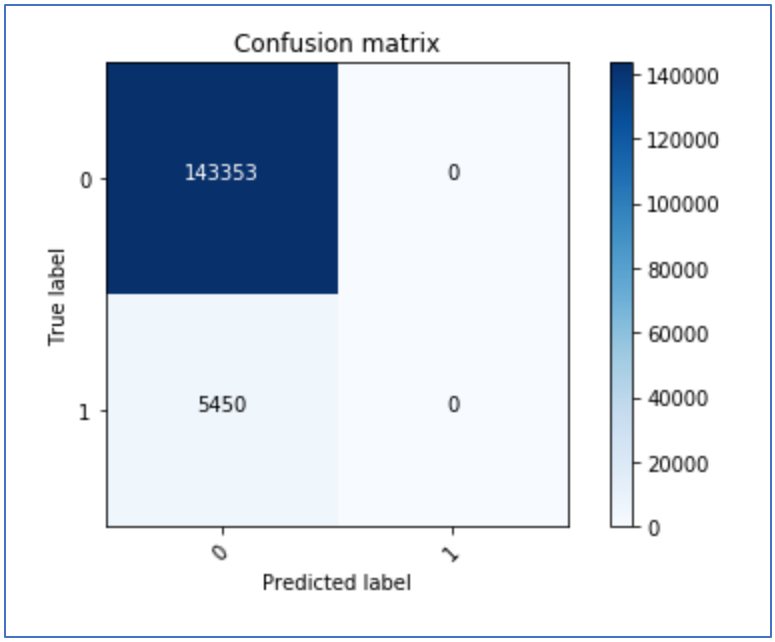


Figure 12: Confusion matrix for Random Forest Version 1

Thus, this model is predicting every record as 0.

Version 2

In the next attempt I did some additional pre-processing as follows:

1. Under sampled the data as shown below:

```
Rate to undersample records with target=0: 0.34043569687437886
Number of records with target=0 after undersampling: 195246
```

2. Dropped 'ps_ind_10_bin', 'ps_ind_11_bin', 'ps_ind_12_bin', 'ps_ind_13_bin', 'ps_reg_03', 'ps_car_03_cat', 'ps_car_05_cat','ps_car_14' as these columns either had large number of null values or low variance

Model parameters were kept the same.

Score obtained using this model: 0.25915

Version 3

In the next attempt I did not drop the columns and kept everything like attempt 2.

Score obtained using this model: 0.26026

Confusion matrix for both attempt 1 and 2 was same as below:

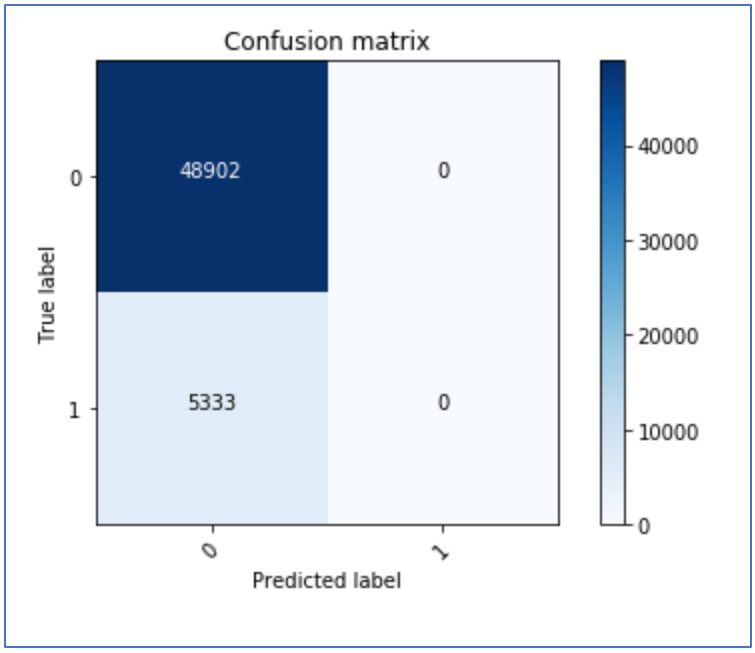


Figure 13: Confusion matrix for Random Forest Version 3

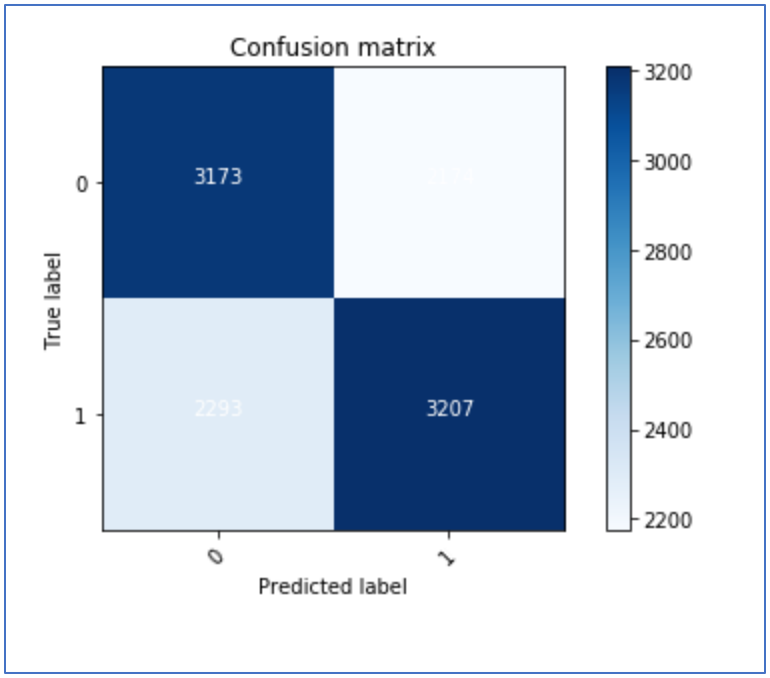
Version 4

Changed the under-sampling rate as show below:

```
Rate to undersample records with target=0: 0.037826188541597645
Number of records with target=0 after undersampling: 21694
```

Score obtained using this model: 0.24853

Clearly, increasing the under-sampling rate is not improving the model performance.




```
Confusion matrix, without normalization
[[3173 2174]
 [2293 3207]]
```

Figure 14: Confusion matrix for Random Forest Version 4

Here, the model is not predicting all as one or zero but still the score obtained is not improving. So, results of confusion matrix may be misleading.

I tried different undersampling rates, but the best score of 0.26302 was obtained for:

```
Rate to undersample records with target=0: 0.7186975822903553
Number of records with target=0 after undersampling: 412186
```

XGBoost

Parameters used were based on kernel by Andy Harless [9].

- 'min_child_weight': 10.0,
- 'objective': 'binary:logistic',
- 'max_depth': 7,
- 'max_delta_step': 1.8,
- 'colsample_bytree': 0.4,
- 'subsample': 0.8,

- 'eta': 0.025,
- 'gamma': 0.65,
- 'num_boost_round': 700

Version 1

For using the xgboost model following preprocessing was performed:

1. Undersampling of records
2. Filling missing values with median values
3. Creating dummy variables for categorical data
4. Divided the training data into 70/30 train test split

Score obtained with this model: 0.26873

Version 2

In the second version I dropped columns starting with “ps_calc_” as they have little variance.

Score obtained with this model: 0.27359

Version 3

In the next version instead of splitting the data into train and test data, I used StratifiedKFold package to do cross validation in order to get the best model.

I varied the fold values from 3 to 5.

Best score obtained with this model was with 5 folds: 0.28177

Challenges Faced

Data exploration and implementation was not that challenging as there are lots of kernels shared by other users on Kaggle for this competition. Kernels are coding, analysis and collaboration product on Kaggle which allows users to run their scripts online. Users can collaborate, and they can share these scripts and results with others.

The only problem was choosing the good ones and building upon them. For data exploration and visualization, a tutorial on plot.ly was very useful and I based my results on them [5]. Random Forest classifiers are easy to implement following the documentation on sklearn. However, using XGBoost was a bit challenging as it requires it inputs in matrix format and there are many parameters that need to be set to get the best results. But XGboost documentation [7] and a kernel by Andy Harless [7] helped me navigate this learning curve.

Most of the jupyter notebooks for this project were run on kaggle’s kernel only but for k-fold cross validated XGBoost model running time exceeded 1-hour limit of Kaggle for running any kernel. So, I had to download the notebooks and run them on my machine.

4.Results

The results of both the models can be summarized as below:

Model	Version	Score	Under sampling	Dropping columns with considerable number of null values	Dropping ps_calc columns	Cross Validation
Random Forest	1	0.26504				
	2	0.25915	✓	✓		
	3	0.26026	✓			
	4	0.26302	✓			
	5	0.26054	✓		✓	
XGBoost	1	0.26873	✓			
	2	0.27359	✓		✓	
	3	0.28177	✓		✓	✓

Thus, following refinements worked well:

1. Under sampling the data to improve the balance of classes.
2. Dropping columns of type ps_calc as they were not correlated with the target value.
3. Cross validation technique to tune the model.

Robustness of models to new data or changes in data is always a concern as in the end it is all about bias-variance tradeoff and overfitting can awlays occur. Random Forest and XGBoost models obtained by this analysis can be considered as robust due to the following reasons:

1. Out of Bagging(oob_score) for random forest was set to true during training.
   The RandomForestClassifier is trained using *bootstrap aggregation*, where each new tree is fit from a bootstrap sample of the training observations $z_i = (x_i, y_i)$. The *out-of-bag* (OOB) error is the average error for each z_i calculated using predictions from the trees that do not contain z_i in their respective bootstrap sample. This allows theRandomForestClassifier to be fit and validated whilst being trained. [14]
   The out of bag error for version 4 of random forest model came out to be 0.101 which is pretty good.

```
oob_error = 1 - RF_model_cat.oob_score_
print('Out of Bag Error: %.3f' % oob_error)

Out of Bag Error: 0.101
```

Figure 15: Out of bag error for Random Forest Version 4 model

2. XGBoost model version 3 is the best model obtained and it was trained using 5-fold cross validation technique which divides the training data into 5 folds and then uses 4 folds to train a model and then uses the fold which was not used in training to test it. This

keeps overfitting under check as the whole data is not being used for training and a part of data is being used to see the performance on unseen data.
For the XGBoost model "feval" parameter was set in order to use gini coefficient as an evaluation function in each iteration. Also, maximize was set to True to tell XGB that higher metric is better. Output generated while training showed the gini coefficient for each 100th iteration and the best iteration of each fold. Output for best iteration of each fold is summarized below:

Fold	Valid-gini
1	0.27649
2	0.282748
3	0.286634
4	0.290524
5	0.278046

The final score obtained on test data provided by Kaggle was 0.28177. Thus, the scores obtained during training are quite close to the final score obtained. Thus, model is robust for unseen data.

5. Conclusion

Random Forest and XGBoost models were used to create submissions for the competition. XGBoost model cross validated with 5 folds and other tunings helped me achieve a score between 0.28 – 0.29 which was my goal.

Some amount of feature engineering can help improve the score more. Also, neural network models can also be used to achieve a better score.

This project helped me understand intricacies involved when handling classification problems with highly imbalanced data. Also, it was a good hands on experience to make use of the most popular model currently – XGBoost.

Bar plot and scatter plot for feature importance for Random Forest version 4 model are as shown below:

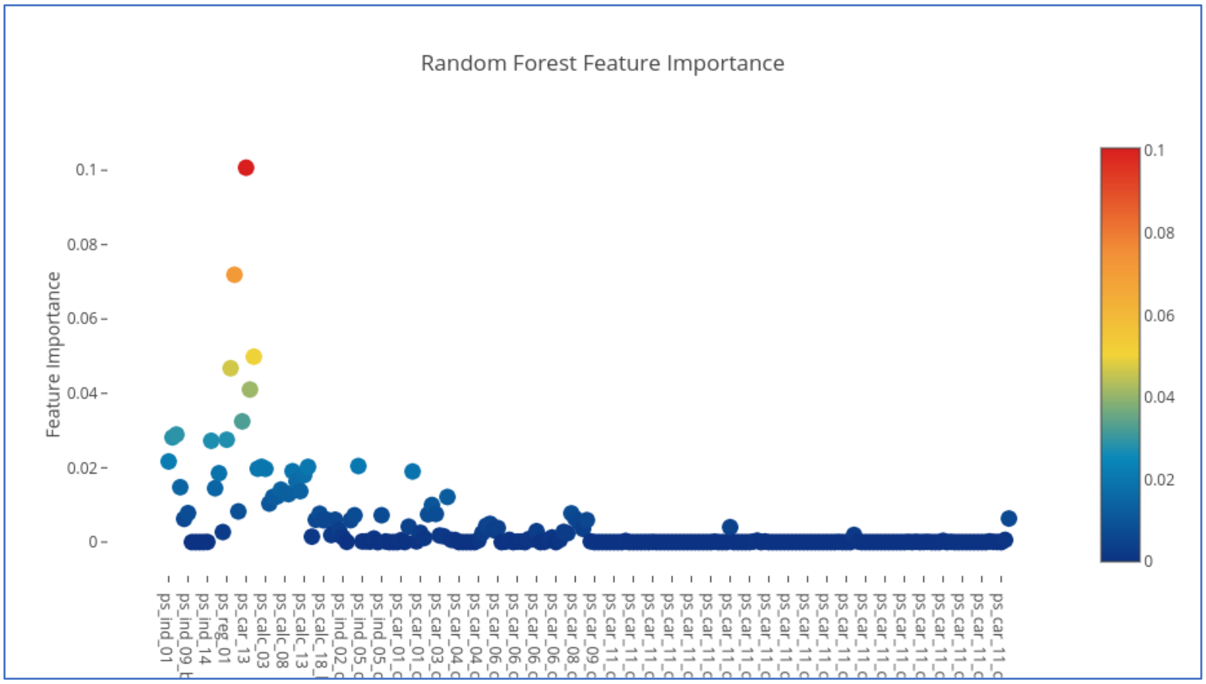


Figure 16: Scatter Plot for feature importance of random forest model

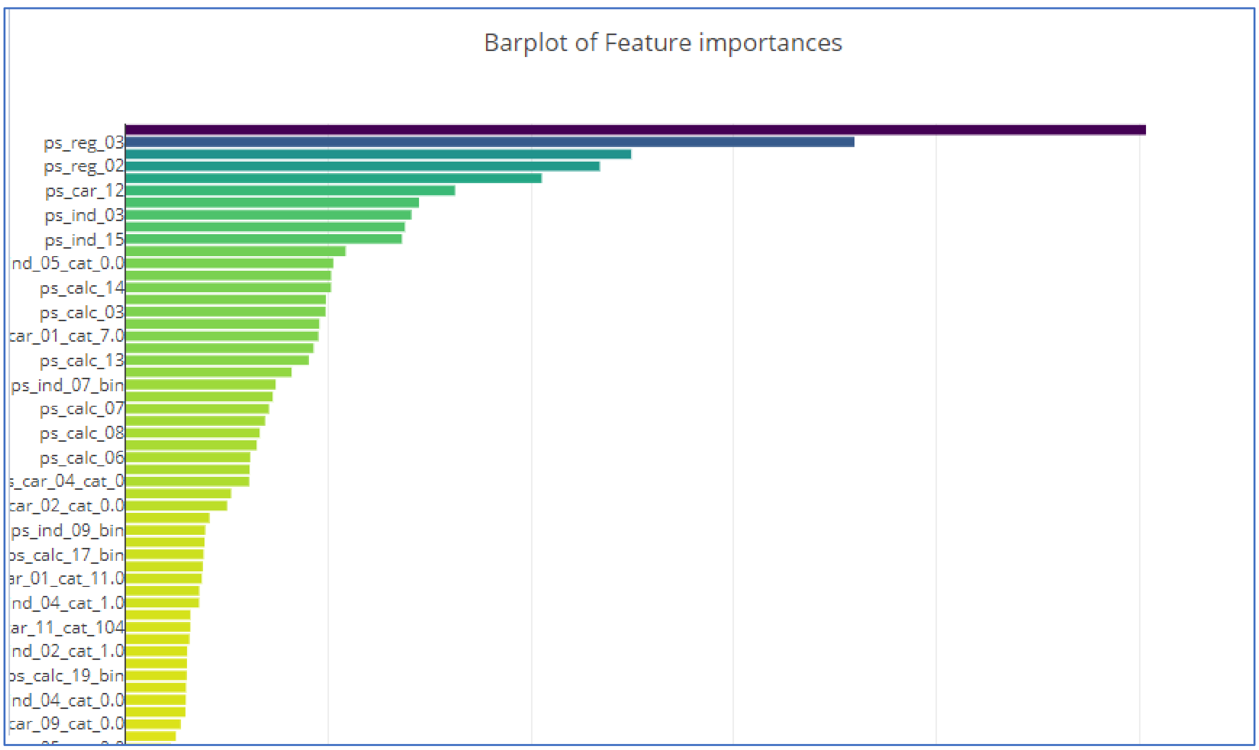


Figure 17: Bar Plot for feature importance of random forest model

There are certain features like ps_reg_03, ps_reg_02 and ps_car_12. However, not much insight can be gained about why these features may be important as description of features is not provided.

Maximum score achievable using Gini coefficient as a metric is 0.5 and even the winning model could achieve a score of 0.29698 only. This indicates that this is a very tough classification problem and a PCA analysis based on a kernel [16] by Kaggle user Tilli shows why this is such a difficult problem.

Principal Component Analysis identifies the combination of components (directions in the feature space) that account for the most variance in the data [17]. This is useful for visualizing high dimensional data using only components that account for most of the variance in data. Principle component analysis was run on training data and dummy variables were created for categorical features. Scatter plot of training data projected on 1st and 2nd Principal components that explain 7.4% and 6.7% of variance is as shown below with target class 0 in blue and 1 in red.

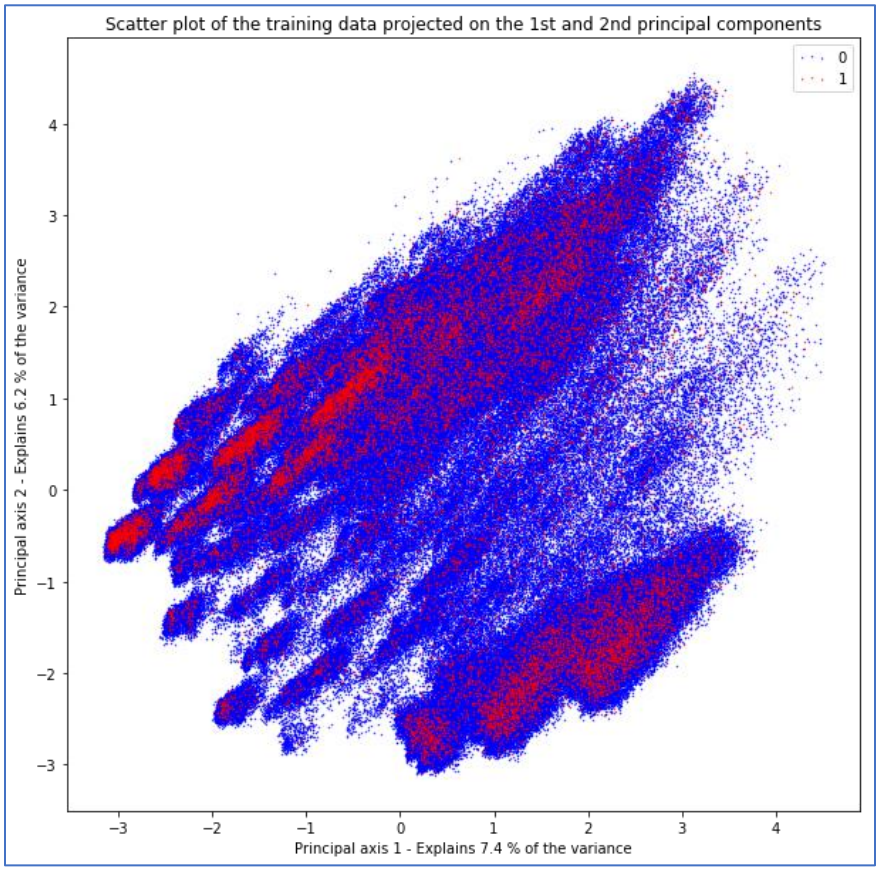


Figure 18: Scatter plot for training data projected on 1st and 2nd Principal Component Analysis

It is visible from this plot that there are very few clearly defined clusters of 1s and there is no clear separation between 0s and 1s. Thus it can be concluded that a score in range of 0.28 - 0.29 is a good score for such a tough classification problem.

References

[1] https://www.pega.com/predictive-analytics-for-insurance

[2] https://blogs.sap.com/2017/08/08/predictive-analytics-impact-on-the-insurance-industry/

[3] https://www.predictiveanalyticstoday.com/predictive-analytics-insurance/

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

[5] https://www.kaggle.com/arthurtok/interactive-porto-insights-a-plot-ly-tutorial

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

[7] http://xgboost.readthedocs.io/en/latest/model.html

[8] https://www.kaggle.com/bertcarremans/data-preparation-exploration

[9] https://www.kaggle.com/aharless/xgboost-cv-lb-284

[10] https://en.wikipedia.org/wiki/Gini_coefficient

[11] https://www.kaggle.com/cpptz/gini-coefficient-an-explanation-with-math

[12] https://en.wikipedia.org/wiki/Boosting_(machine_learning)

[13] https://www.analyticsvidhya.com/blog/2015/11/quick-introduction-boosting-algorithms-machine-learning/

[14] http://scikit-learn.org/stable/auto_examples/ensemble/plot_ensemble_oob.html

[15] http://blog.kaggle.com/2016/07/08/kaggle-kernel-a-new-name-for-scripts/

[16] https://www.kaggle.com/tilii7/dimensionality-reduction-pca-tsne

[17] https://en.wikipedia.org/wiki/Principal_component_analysis