IIITB ML Project: PS Safe Driver Prediction

InClass competition for AI511 course 2020 at IIIT Bangalore

Akshath Kaushal (IMT2018501)
Shathir Hussain (IMT2018070)
Sree Harsha Koyi (IMT2018512)

***Abstract*— This document is the final result of the collaboration by the members of the team YSoSerious on the semester project of the AI-501 Machine Learning course offered by IIIT Bangalore.**

***Keywords*— Normalized gini, RandomizedSearchCV, one hot encoding, Class imbalance, Classification**

I. Introduction

Nothing ruins the thrill of buying a brand new car more quickly than seeing your new insurance bill. The sting's even more painful when you know you're a good driver. It doesn't seem fair that you have to pay so much if you've been cautious on the road for years.

Porto Seguro, one of Brazil's largest auto and homeowner insurance companies, completely agrees. Inaccuracies in car insurance company's claim predictions raise the cost of insurance for good drivers and reduce the price for bad ones.

II. Problem Statement

In this competition, you're challenged to build a model that predicts the probability that a driver will initiate an auto insurance claim in the next year. While Porto Seguro has used machine learning for the past 20 years, they're looking to Kaggle's machine-learning community to explore new, more powerful methods. A more accurate prediction will allow them to further tailor their prices, and hopefully make auto insurance coverage more accessible to more drivers.

III. Dataset

The data set used is a fraction of the public dataset that is available on the official competition's page on Kaggle.

The dataset contains the target column which predicts whether the driver will file a claim or not as well as other columns/features such as ps_ind_01, ps_car_07_cat, ps_calc_06 etc. which are to be used for training.

Also, various features have been labelled for convenience in the following way:

- bin: binary features
- cat: categorical features

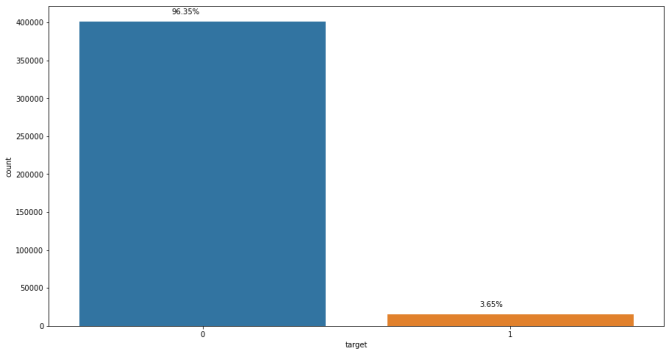
Rest with no particular label are the continuous or ordinal features.

In total, there are 59 columns, 416648 rows in the training dataset and 58 columns, 178564 rows in the test dataset.

Missing values are present in the dataset, but they have been replaced by -1 instead of null for convenience.

IV. Exploratory Data Analysis

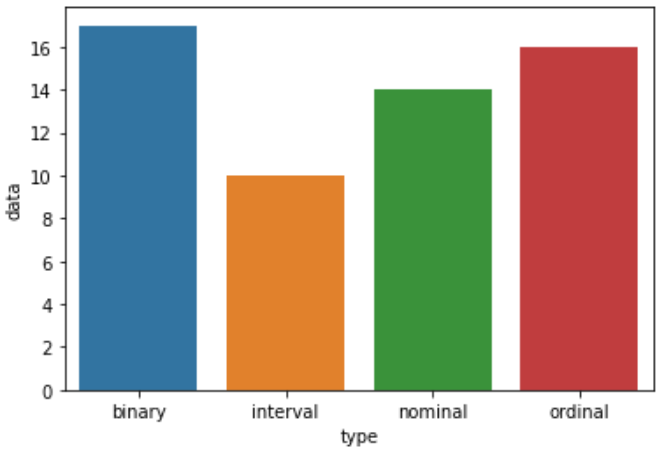
The first step we took was getting to know the target column. For this, we plotted a simple bar graph of the two classes, 1: driver claims and 0: driver doesn't claim.



This clearly portrayed the imbalance in the data, hence for this classification problem, we had to follow techniques that took care of this imbalance and generate predictions accurately.

After this, we started looking at the different features.

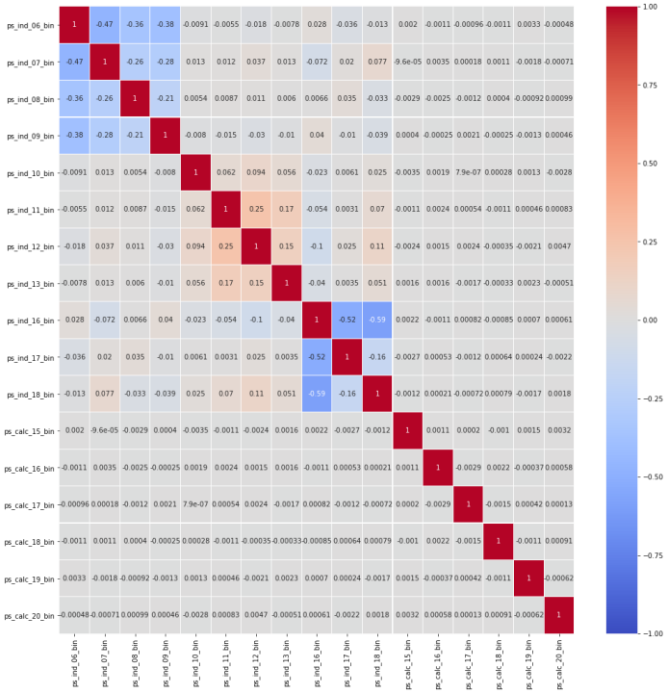
The way we approached the feature exploration was very simple. First we divided the features into three parts, namely categorical, binary and numerical.



This showed that there are 17 binary, 14 nominal/ categorical, and 16 + 10 numerical columns.

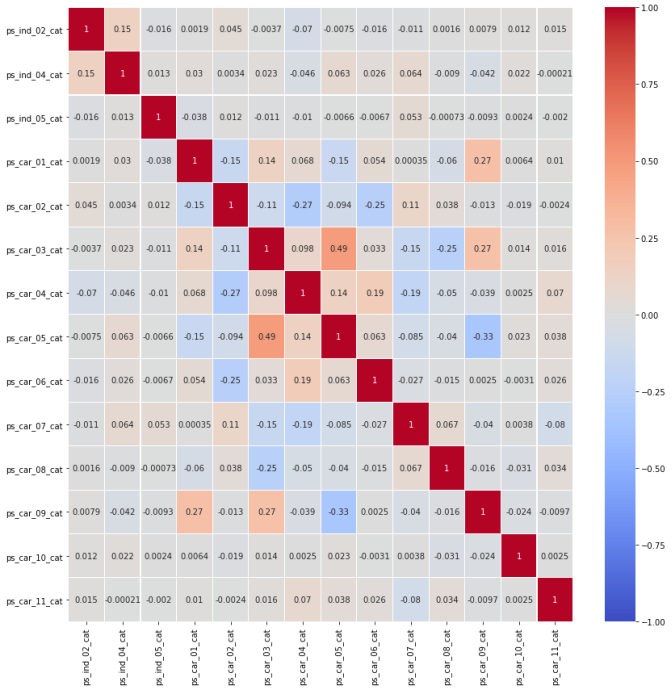
After this, we approached each of the three parts separately. First, we plotted the heat plots for each type of feature to see the correlations. After this, we separately plotted each feature to see the distributions.

1. Binary columns

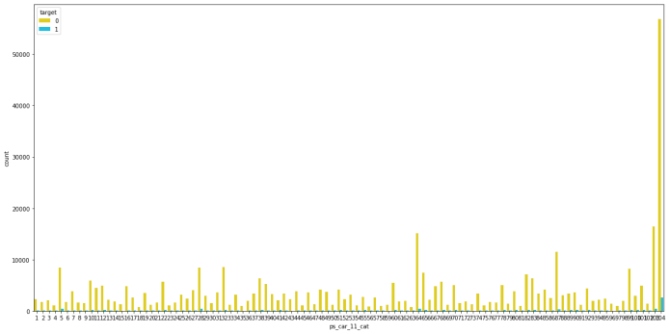


This clearly showed the absence of any major correlation between binary features. Hence, we didn't have to remove any columns and we proceeded.

2. Categorical columns



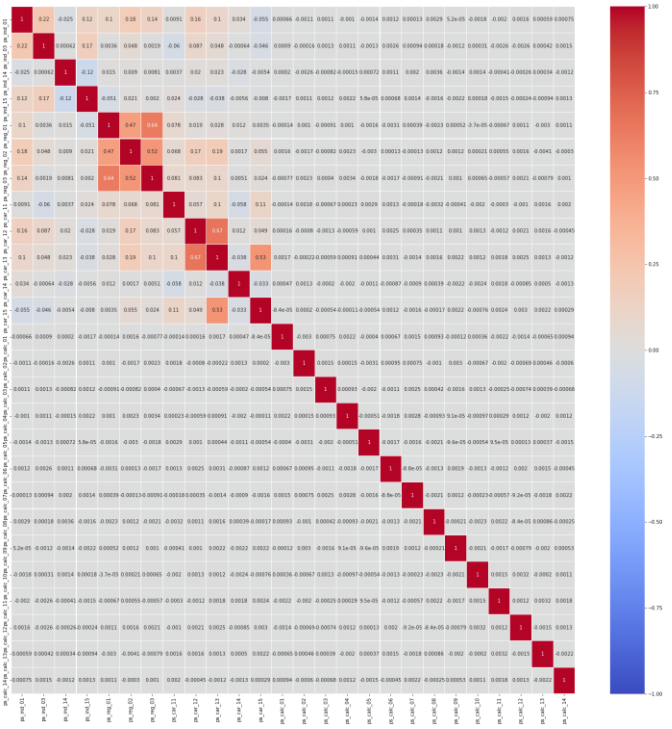
As in the case of binary columns, there wasn't any significant correlation, so we proceeded further.



These were the main insights and some methods we thought of:

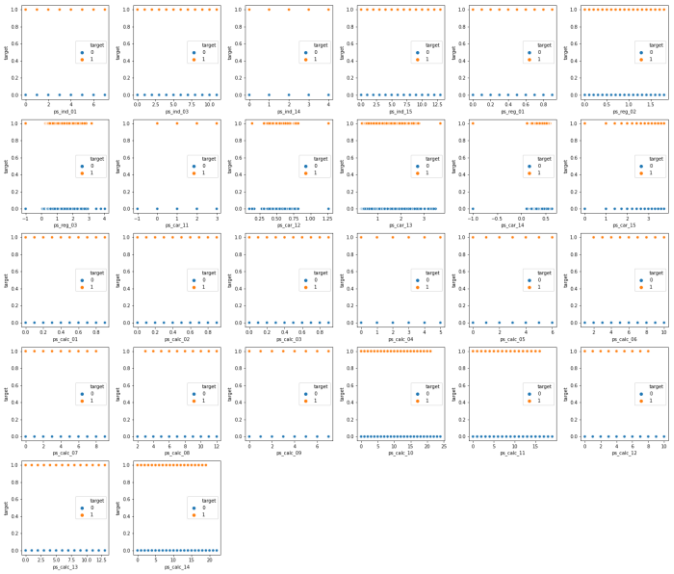
- From these plots, we saw that except a few features (that had “calc” in their column names), most of them were dominated by a single value.
- Some features like 'ps_ind_05_cat' and 'ps_car_04_cat' mostly consists of a single value. Therefore the mode of these features can be used for filling their missing values
- We plan to replace the -1s with either mean, median or mode in categorical values.
- In ps_car_03_cat and ps_car_05_cat, there is a higher number of -1s, so we can either drop the column or treat -1 as a separate category.
- After all this, we will go for techniques like OHE and Label encoding etc.

3. Numerical columns

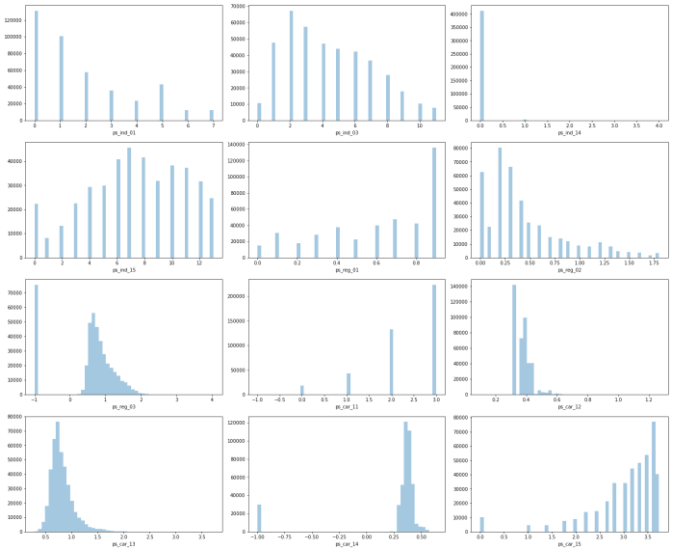


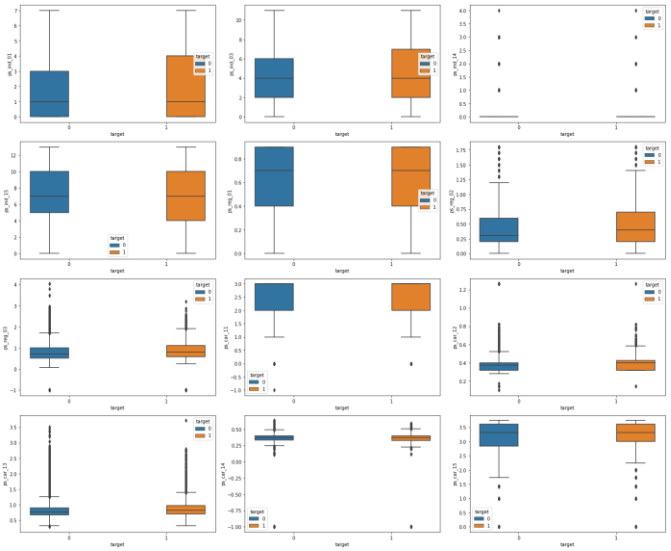
Here also, there was no significant correlation and we proceeded further.

In the numerical columns especially, we found some columns had the word “calc” in their names. Though we did think that this word might stand for “calculated”, we initially didn’t pay enough attention and moved on. Also, since the explanation of each feature has not been disclosed by the company, there was not much we could find without exploring.
Hence, we decided to first check the data by scatter plots.



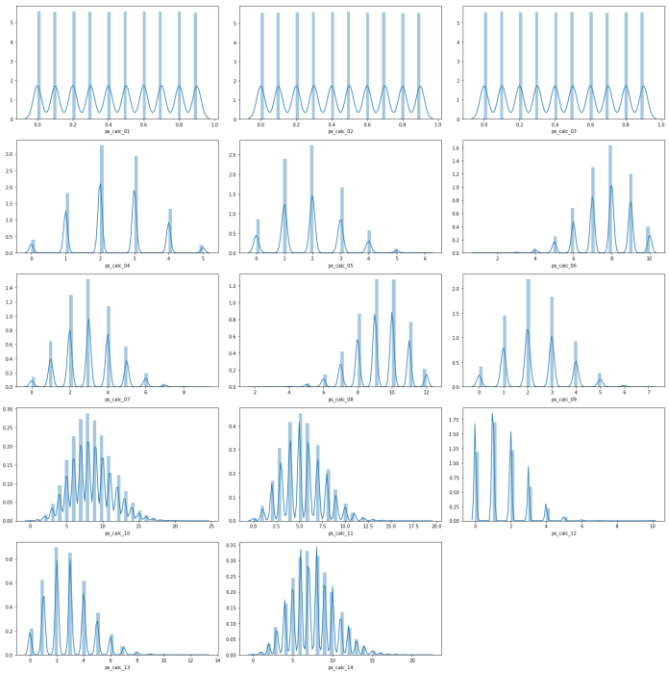
After this, we were still not convinced. So we divided the numerical columns into further two parts, one comprising of “calc” features, and the other without them.
First we plotted distplots and box plots of “non-calc” features.

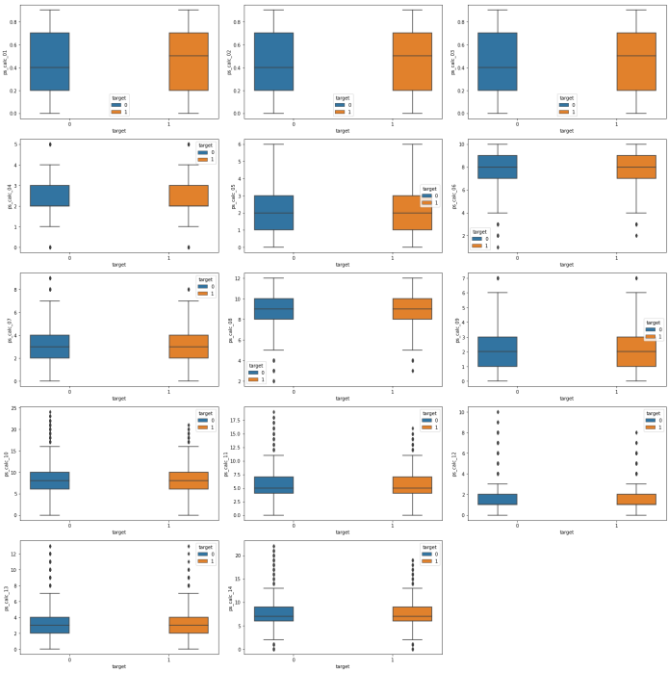




This showed that the data was somewhat skewed and had a few outliers.

After this, we moved to “calc” columns.





From these plots, we were almost convinced about the nature of “calc” features. We initially suspected these features to be calculated and these plots confirmed our suspicion. Since these features had a normal distribution and the box plots for both classes (claim/not claim) were almost similar, it was absurd to predict using these feature. They were essentially noise for the model and did not provide any information. Nevertheless, we did generate predictions using these features and the results were worse than those obtained after removing them. So, we removed these features.

V. LIGHTGBM

At this point, after completing the EDA, we decided to apply LightGBM to the dataset. This would generate predictions as LightGBM does not require any pre-processing. Also, another benefit of using LightGBM is that it has a function that returns the importance of each feature. This would further help in cleaning the dataset.
Here are some benefits of LightGBM:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

Hyperparameter tuning:

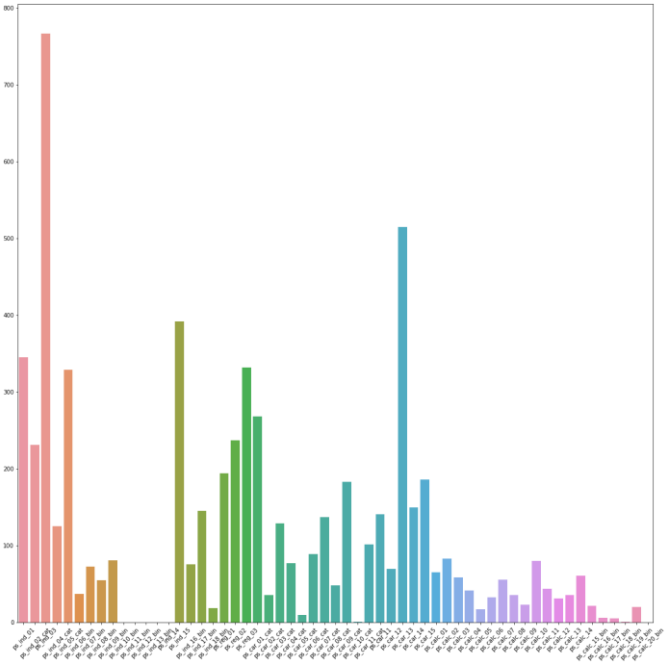
These were the hyperparameters we tuned for LightGBM:

- Learning rate
- Number of estimators
- reg_alpha and reg_lambda
- Number of leaves
- The sample training fraction

We used RandomizedSearchCV from sklearn's model_selection library in order to determine the appropriate hyperparameters. These were the final hyperparameters we used:

Hyperparameter	**Optimum value**
n_estimators	700
learning_rate	0.01
reg_alpha	4
reg_lambda	4
num_leaves	10
subsample	0.8

Here are the feature importance obtained from the tuned LightGBM.



As seen from the plot, the columns ps_ind_10_bin, ps_ind_11_bin, ps_ind_12_bin, ps_ind_13_bin, ps_ind_14 and ps_calc_20_bin have 0 feature importance. So we removed these columns.

VI. CATBOOST

CatBoost is an extremely robust algorithm. It does not require any specific pre-processing and yet can produce flawless results. Due to this reason, we thought of applying it before pre-processing the data.
Even though CatBoost can give good results without any hyperparameter tuning, yet we tuned it a little using RandomizedSearchCV, as before.
These are the final hyperparameters we used

Hyperparameter	Tuned Value
n_estimators	1000
learning_rate	0.03
depth	7

VII. PRE-PROCESSING OF DATA

1. Imputing

Since the dataset has missing values (even though they are filled with -1s), we imputed the different types of features in the following way:

- Numerical columns:
  ps_reg_03, ps_car_14 were imputing using mean as the strategy.
  Ps_car_11 had only 1 value missing, so it was imputing using mode.

- Categorical columns
  ps_ind_02_cat, ps_ind_04_cat, ps_ind_05_cat, ps_car_01_cat, ps_car_02_cat, ps_car_07_cat, ps_car_09_cat were imputed using mode as a strategy.
  ps_car_03_cat and ps_car_03_cat were left as it is (due to less number of missing values, -1 was treated as a separate category).

2. Encoding

We did one hot encoding for categorical variables. The binary columns already had digits**.**

3. Upsampling

Since the data was imbalanced, we decided to balance the dataset. For this, we could do either oversampling or undersampling. Since the imbalance is large, undersampling would lead to a lot of information loss, so we stuck with oversampling and decided to control the overfitting in some way. We used sklearn's resample from the utils library.
After this, we had 602141 rows and 207 columns in the training dataset.

4. Standardization

In this process, mean of each column is subtracted from each entry and the result is divided by the standard deviation of the entire column.
This process is a pre-requisite of many Machine learning algorithms.
Also, we could have taken the min-max norm, yet we stuck to this as this provided better results.
We used StandardScaler from the sklearn's preprocessing library.

VIII. EVALUATION CRITERIA

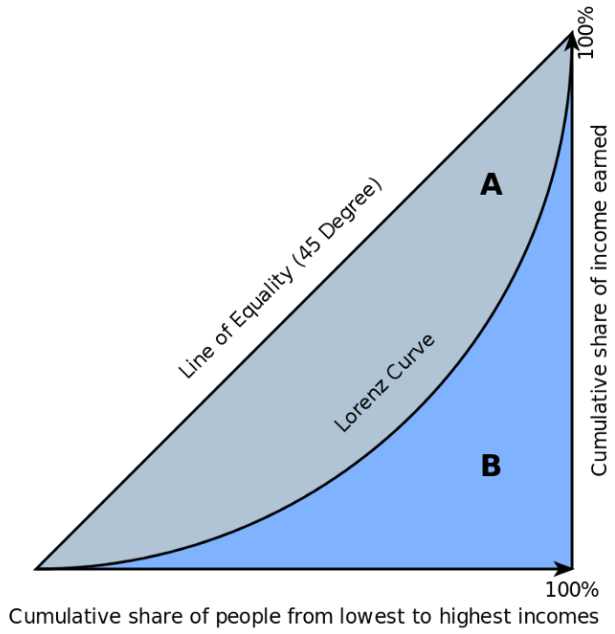
The evaluation criteria of the competition was normalized gini index.

The Gini Coefficient or Gini Index measures the inequality among values of a variable. Higher the value of an index, more dispersed is the data. Alternatively, the Gini coefficient can be looked like half of the relative mean absolute difference.

It is related to area under curve (AUC) as

$$Gini = 2*AUC - 1$$

The Gini coefficient is usually defined mathematically based on the Lorenz curve, which plots the proportion of the total income of the population (y-axis) that is cumulatively earned by the bottom x% of the population (see diagram). The line at 45 degrees thus represents perfect equality of incomes. The Gini coefficient can then be thought of as the ratio of the area that lies between the line of equality and the Lorenz curve (marked A in the diagram) over the total area under the line of equality (marked A and B in the diagram); i.e., G = A / (A + B). It is also equal to 2A and to 1–2B due to the fact that A + B = 0.5 (since the axes scale from 0 to 1).



IX. TRAINING MODELS

After preprocessing the data, we started training models.

1) Linear regression

Initially we used a ridge regularized linear regression as a base model. We did not get very accurate predictions and the maximum score achieved was about 0.19.

2) Stochastic Gradient descent

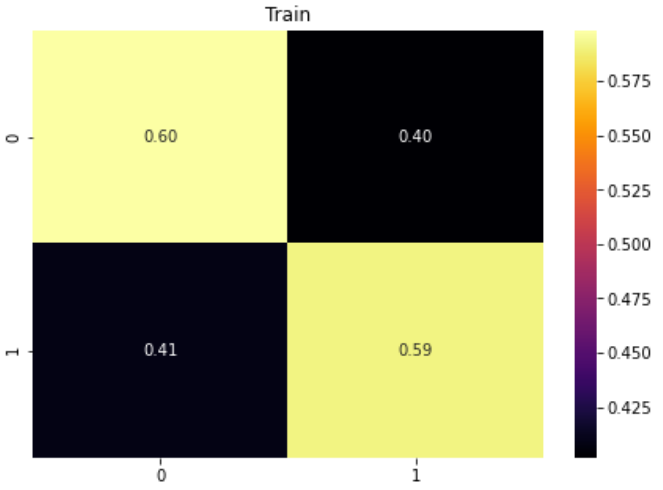
We used the SGDClassifier available in sklearn.

The SGDClassifier applies regularized linear model with SGD learning to build an estimator. The SGD classifier works well with large-scale datasets and it is an efficient and easy to implement method.

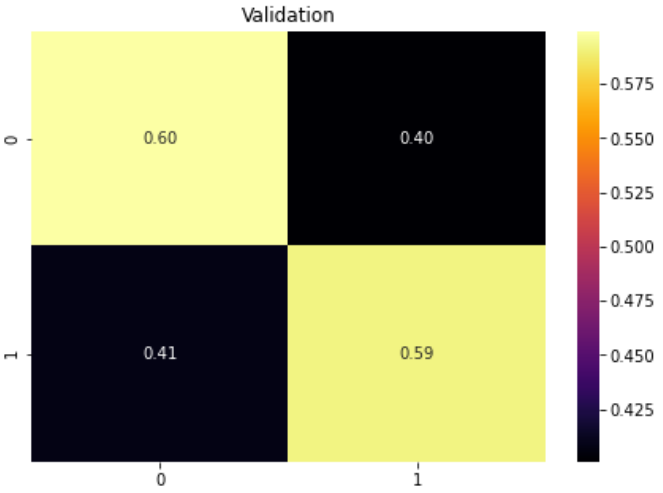
Not much tuning was required. Using the loss function as log loss and class_weight to balanced, we chose the optimum learning rate to achieve a score of about 0.26.

Here is the performance evaluation of SGDClassifier:

- Training data



- Validation data



3) SVM

The objective of the support vector machine algorithm is to find an appropriate hyperplane in an N-dimensional space (N, the number of features) that best classifies the data points.

SVM uses mathematically defined functions called kernels that take data as input and transform it to compute on it.

There are various types of kernels namely linear, non-linear, polynomial, radial basis function (rbf) and sigmoid.

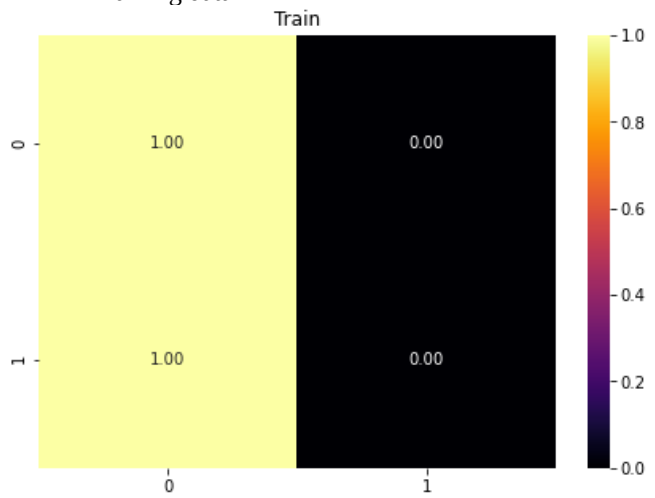
We used rbf kernel in our notebook, as it gave the best answer in our case.

$$k(\mathbf{x_i}, \mathbf{x_j}) = \exp(-\gamma \|\mathbf{x_i} - \mathbf{x_j}\|^2)$$

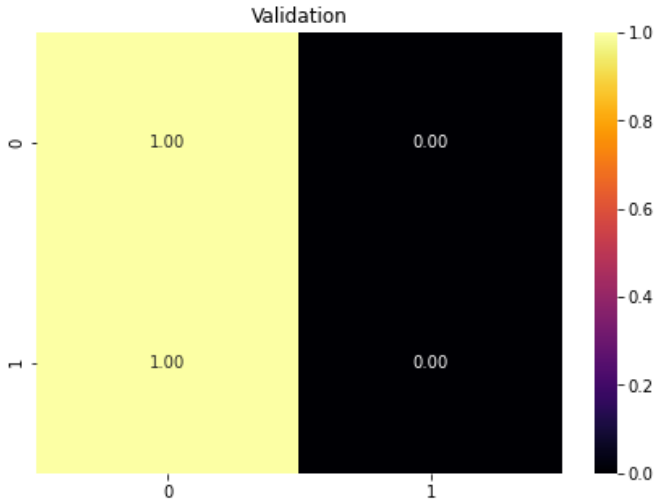
Gaussian radial basis function (RBF)

But due to complexity of the dataset, SVM was taking too much time to run, so we used less number of iterations and the result was not good.
Here is the performance analysis of SVM:

- Training data



- Validation data



As seen from the plots, the result is not of any help.

4) Gaussian Naïve Bayes

It is the extension of the generic Naïve Bayes Classifier.
This form is the easiest to work with and can generate pretty accurate results. Also, this classifier works well with continuous features. The likelihood function that GNB uses is:

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Since our dataset had continuous features, we stuck to this classifier and got a score of approx. 0.21 upon cross validation.

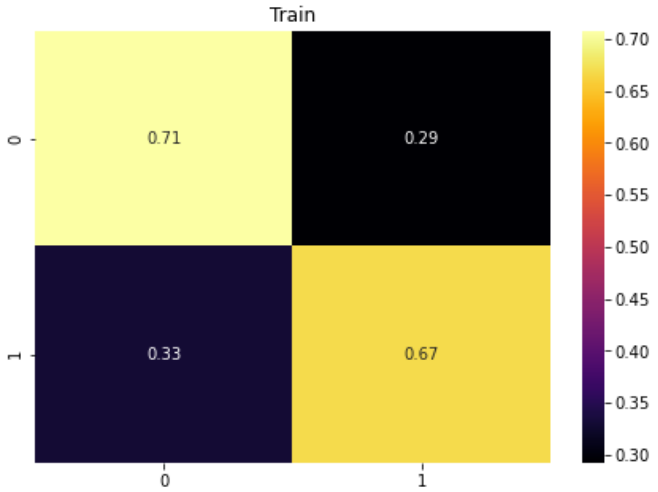
5) Decision Tree

Decision Tree is a supervised learning algorithm which uses tree-like structure in order to classify the test data. It can be used for both classification and regression problems.
At every step, decision tree calculates the information gain and then splits according to the feature that provides the highest information gain.
Also, decision trees are prone to overfitting, so we carefully tuned our model and it provided pretty good results.
We tuned the model with a randomised search.
Here are the hyperparameters we tuned along with their final values:

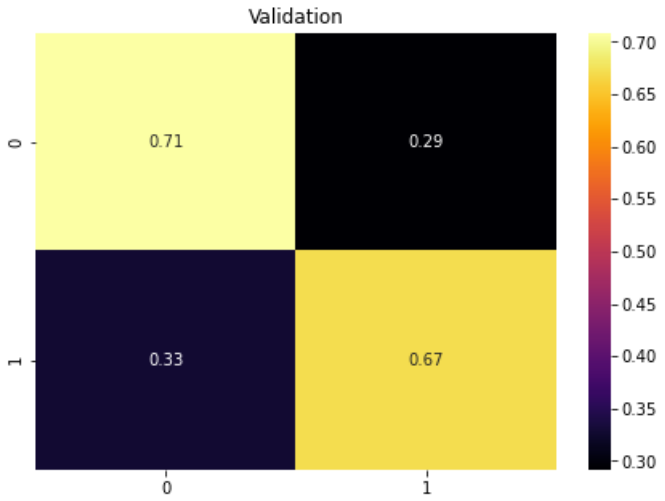
Hyperparameter	**Tuned value**
max_features	27
max_depth	15
min_samples_split	5
min_samples_leaf	5
max_leaf_nodes	6000
min_impurity_split	0.05

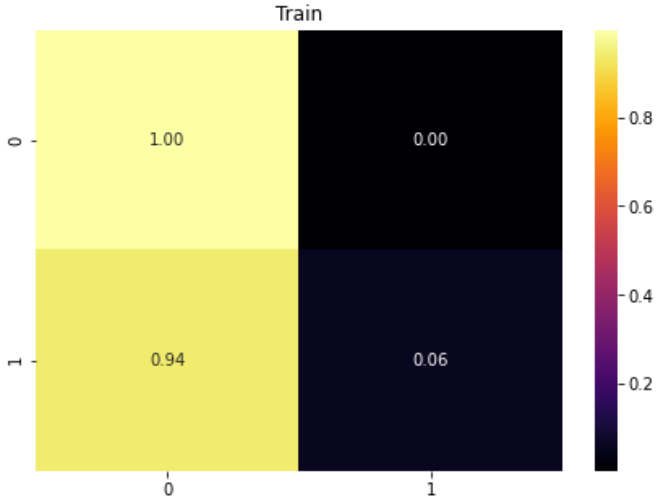
Also, we kept the class_weight to be “balanced” and chose the function to measure the quality of split as gini. Also, the strategy to split was kept as “best”.
Here is the performance analysis:

- Training data



- Validation data





As seen from the results, we get pretty decent results from a single tree. So the results would be good if we implemented tree based algorithms.

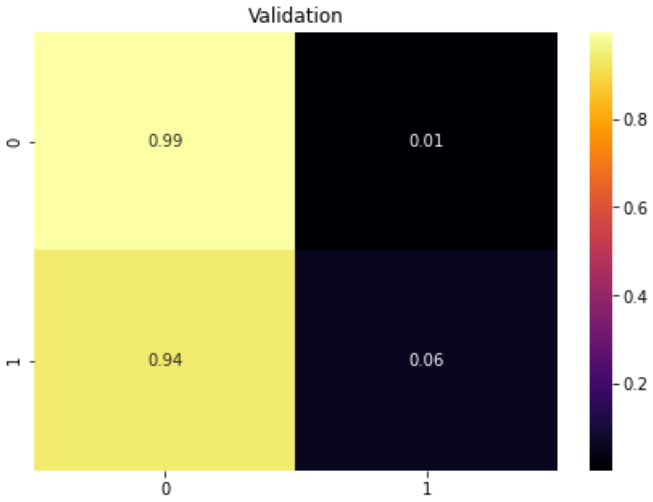
6) Random Forest Classifier

Random forest creates an ensemble (based on divide-and-conquer approach) of numerous decision trees and aggregates the prediction on the basis of majority decision.
This is extremely helpful in preventing overfitting and generalizing the model.
The only issue of RandomForest is that its time consuming. Due to lack of GPU support, we limited the number of cross validations in our implementation, yet it provided pretty good results.
The hyperparameters we tuned were:

Hyperparamter	Tuned value
n_estimators	300
depth	9

Also, the criterion from split was again kept as “gini” and the out_of_bag score for the model was set to True. This helped in further generalizing the model.
The cross validation gini score obtained was around 0.35
Here is the performance analysis:

- Training data



- Validation data

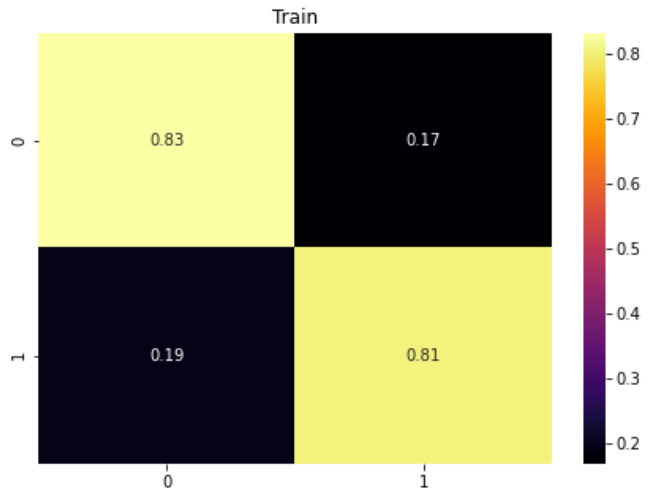
7) Adaboost

Adaboost, or adaptive boosting, is a boosting technique that is used as an ensemble method in machine learning.
In this, the weights are reassigned to each instance, with higher weights to incorrectly classified instances.
In this, the learners grow sequentially.
We defined a base estimator for the Adaboost model as the decision tree that we used previously. By keeping the algorithm “SAMME”, instead of “SAMME_R” (default) and training a couple of hyperparameters, we were able to get a validation score of 0.78

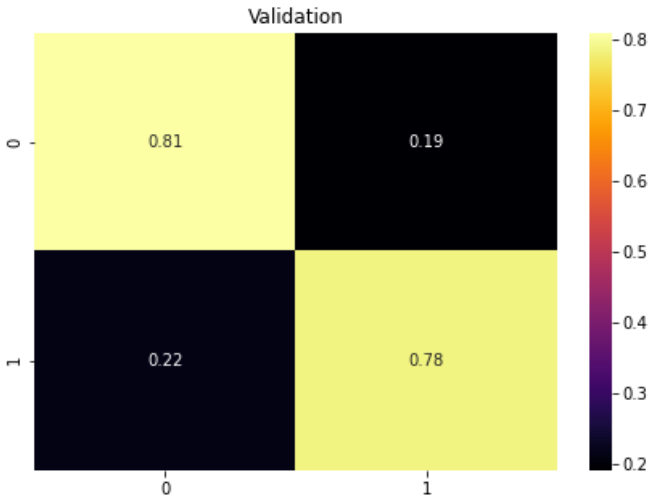
Hyperparameters	Tuned value
n_estimators	1000
learning_rate_	0.01

Here is the performance analysis:

- Training data



- Validation data



8) Gradient Boosting Classifier

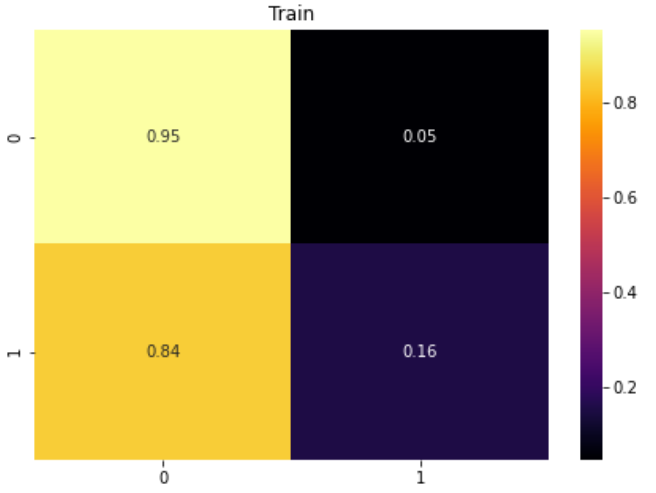
In Gradient Boosting, each predictor tries to improve on its predecessor by reducing the errors. But instead of fitting a predictor on the data at each iteration, it actually fits a new predictor to the residual errors made by the previous predictor.

We used randomized search with a 3 fold cross validation in order to tune hyperparameters. But after a few tweaks, we settled in the following:

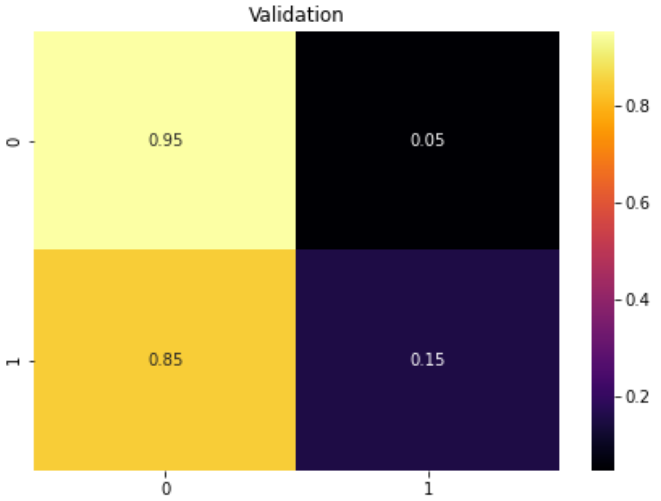
Hyperparameters	**Tuned value**
n_estimators	250
learning_rate	0.1

From these, we were able to get a validation score of about 0.33.
Here is the performance analysis:

- Training data



- Validation data



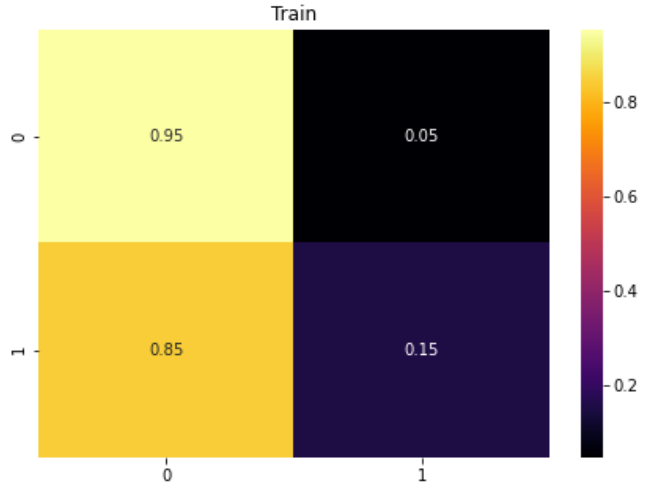
9) XGBoost

XGBoost is an optimized distributed gradient bosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.
We used randomized search with a 5 fold cross validation to determine 6 most useful parameters for out dataset.

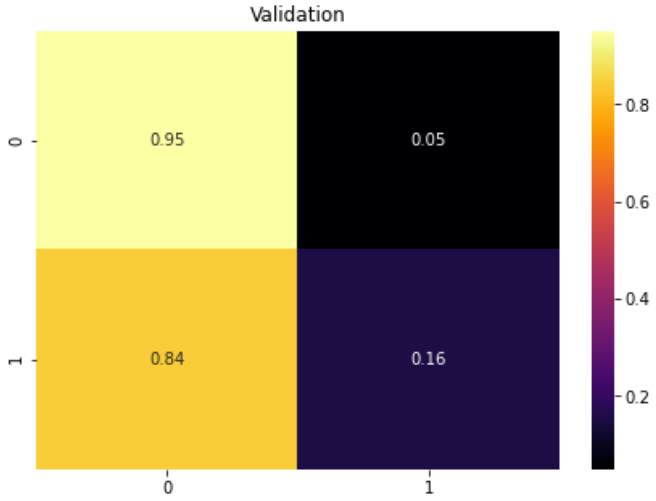
Hyperparameters	Tuned value
n_estimators	200
max_depth	5
learning_rate	0.1
min_child_weight	99
max_leaves	9
reg_lambda	0.5

Also, we kept the sampling method to be gradient_based, i.e. the selection probability for each training instance is proportional to the regularized absolute value of gradients. And the grow policy was set to "lossguide", in which the nodes are split according to highest loss change.
Also, since XGBoost takes time in order to fit on the large datasets, we used the GPU implementation of XGBoost.
We got a validation score of about 0.34.
Here are the performance metrics:

- Training data



- Validation data



X. ENSEMBLING MODELS

1) Combining models

First approach we used was to combine the results of models to get an overall score.
Out of all the combinations we tried, a combination of 20% XGBoost and 80% CatBoost gave the highest score of 0.29161 on public leaderboard and 0.26561 on private leaderboard.
Second submission had a 90% CatBoost and 10% LightGBM. This gave a score of 0.28972 on public leaderboard and 0.26539 on the private leaderboard.

2) Stacking

We tried this approach of stacking using the StackingClassifier available in mlxtend.
Stacking is a way to ensemble multiple models. The point of stacking is to explore a space of different models for the same problem. The idea is that a problem can be attacked with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. So, we can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model. Then we add a new model which learns from the intermediate predictions the same target. This final model is said to be stacked on the top of the others.

XI. SCORES FOR DIFFERENT MODELS

Here are the scores obtained for different models:

Model	Public leaderboard score
LightGBM	0.27895
SGD	0.27632
SVM	-0.02472
Gaussian Naïve Bayes	0.22224
Decision Tree	0.13460
Adaboost	0.27843
Gradient Boosting	0.28580
XGBoost	0.28620
CatBoost	0.28857
Random Forest	0.27267

XII. CONCLUSION

The final model successfully predicts with a gini score of 0.29161 on public leaderboard and 0.26590 on private leaderboard.
There were a lot of possibilities to go about the EDA and modelling. Some of the approaches we took proved fruitful while some did not. Visualization of the dataset enabled us to take an approach that helped us to achieve the score.
Overall there was a lot of learning and certainly an enriching experience that enabled us to push our boundaries of learning.

XIII. FUTURE WORK

One of the ways classification is done is by implementing Neural Networks on the dataset. This would generate the predictions with higher accuracy. Also, some amount of feature engineering can be done. Though this would require further insights on the dataset.
Also, Stacking can be done better by trying out more possible combinations of the models.

XIV. Acknowledgement

We would like to thank Professor G. Srinivas Raghavan, Professor Neelam Sinha and our ML TAs Tejas Kotha, Shreyas Gupta, Tanmay Jain, Vibhav Agarwal, Tushar Anil Masane, Mohd Zahid Faiz, Saurabh Jain, Divyanshu Khandelwal, Bukka Sai Nikhil, Arjun Verma and Amitesh Anand for giving us an opportunity to work on this project and helping us whenever we were stuck with a problem by giving us ideas and showing the way out.
Also, the leaderboard was a great motivation for us and the tough competition enabled us to read and learn various concepts on our own.

XV. References

- Logistic regression with sklearn docs
- LightGBM documentation
- CatBoost documentation
- RandomForest documentation
- Adaboost documentation
- Gradient boosting with sklearn docs
- Decision trees with Analytic Vidhya
- Naïve Bayes documentation
- EDA with seaborn docs
- Matplotlib documentation
- Ensembling techniques by towardsdatascience