“Safe Driver Prediction”

Master of Technology(Integrated)

in

Software Engineering

By

K.A.Tejasri

18MIS0029

S.S.Dheeraj

18MIS0295

Under the guidance of Prof. /

Dr.Subhashini.R

SITE VIT, Vellore.

June, 2021

CONTENTS **PageNo.**

1.Abstract 3

2.Introduction 3

3.Problem Statement 3

4.Dataset 3

5.Algorithms Used 3-4

6.Requirements 4

7.System Architecture and Directory Design 4

8.Flowchart 5

9.Exploratory Data Analysis 5-9

- **9.1 Binary columns**
- **9.2 Categorical columns**
- **9.3 Numerical Columns**

10.Algorithms 9-11

- **10.1 LightBgm**
- **10.2 CatBoost**

11.Evaluation Criteria 11-12

12.Training Models 12-20

- **12.1 Linear Regression**
- **12.2 Stochastic Gradient descent**
- **12.3 SVM**
- **12.4 Decision Tree**
- **12.5 Random Forest Classifier**
- **12.6 Adaboost**
- **12.7 Gradient Boosting**
- **12.8 XGBoost**

13.Combining Models 20

14.Scores For Different Models 20

15.Code Snippets 20-25

16.Conclusion 26

17.Future Work 26

18.References 26-27

1.ABSTRACT:

In this project we are going to take the problem given by brazilian insurance company that to predict the number of auto insurance claims that will be claimed by the customers in the following year by analyzing the data present in the present year. So that we can predict how many are going to claim their insurance and we can also predict the safe driver

2.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

3.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.

4.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.

5.ALGORITHMS USED:

→ CatBoost

→ Decision Tree

→ Gaussian Naïve Bayes

→ Linear Regression

→ Logistic regression

→ SVM

→ Adaboost

→ GradientBoostingClassifier

→ XGBoost

6.REQUIREMENTS:

→ Anaconda IDE

→ Jupiter Notebook

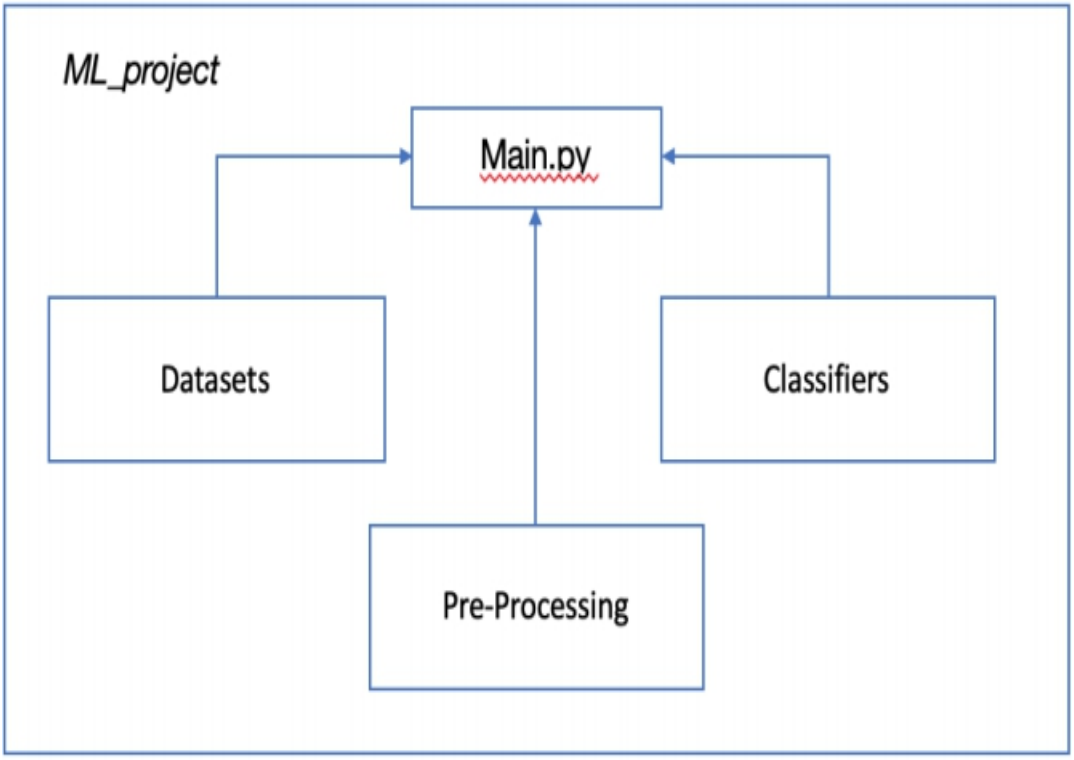
→ Python3

→ CPU: 2 x 64-bit 2.8 GHz 8.00 GT/s CPUs.

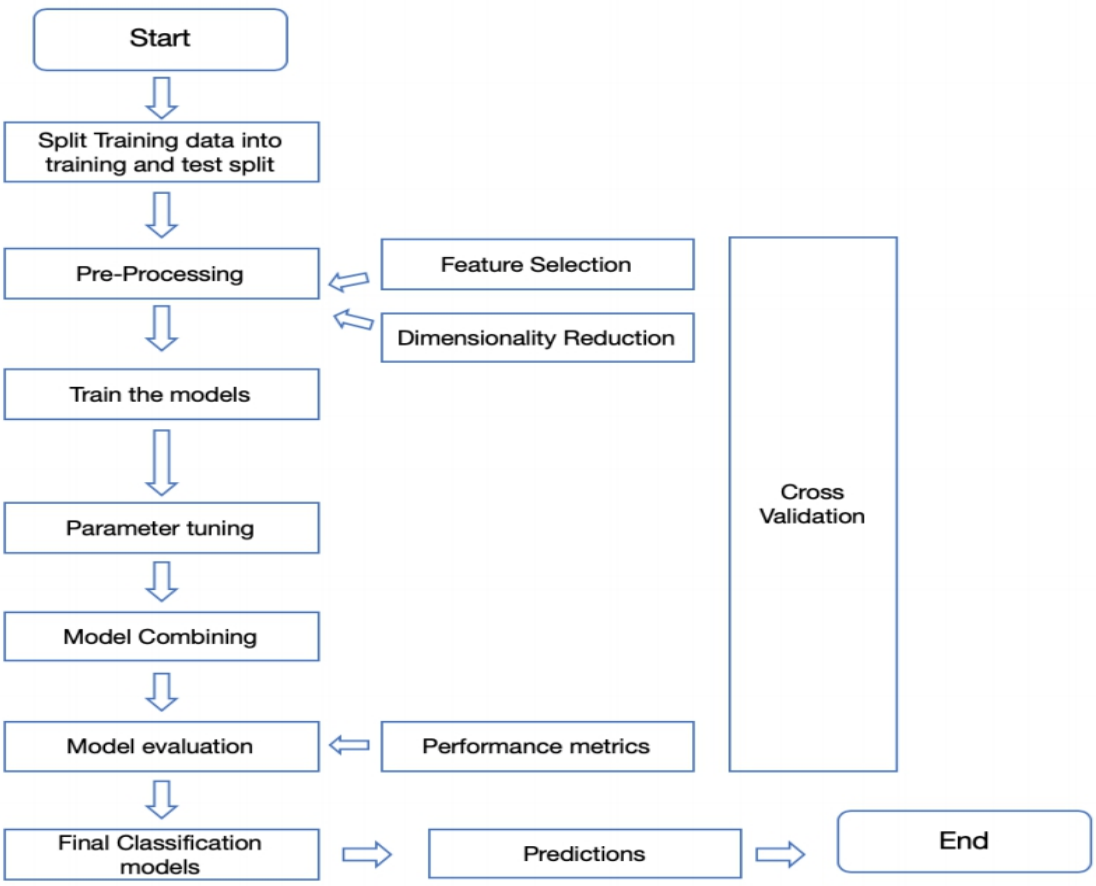
→ RAM: 6 GB (or 4 GB of 1600 MHz DDR3 RAM)

→ Client environment may be Windows, macOS or Linux

7.SYSTEM ARCHITECTURE AND DIRECTORY DIAGRAM:



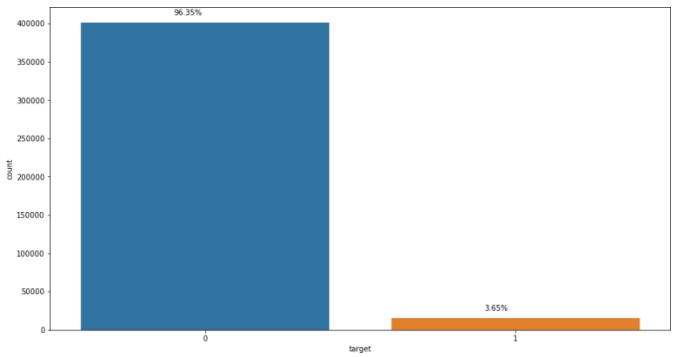
8.SYSTEM DESIGN FLOWCHART:



9.EXPLORATORY DATA ANALYSIS

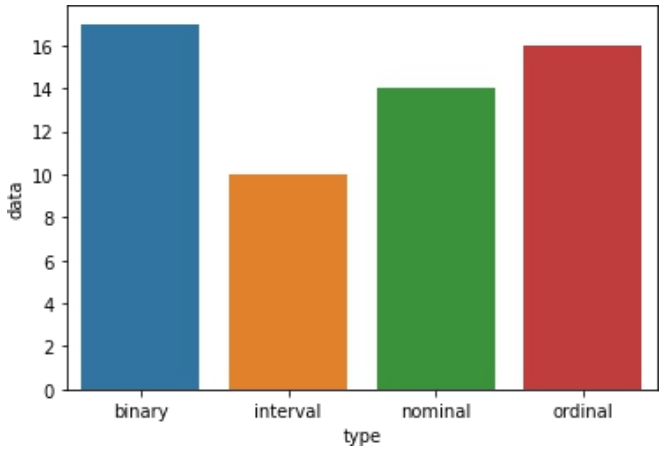
The first step we took was getting to know the target column. For this, we plotted a simple bar graph of the two classes,

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.

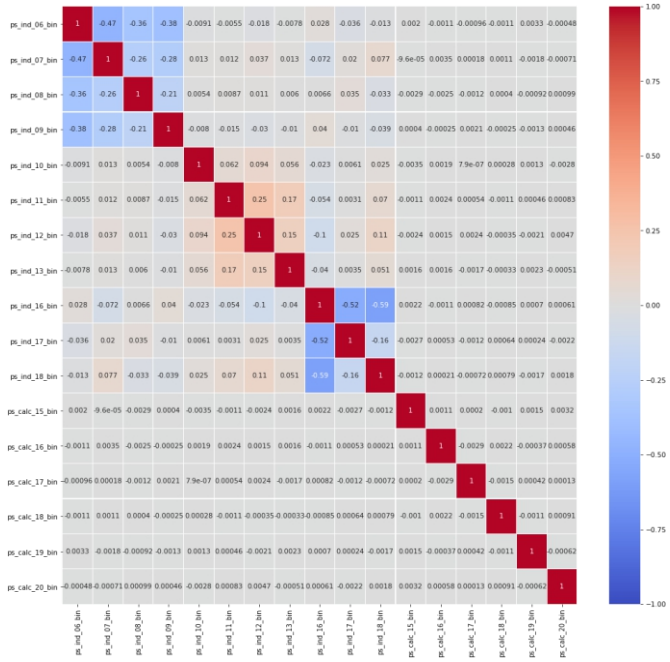
After this, we started looking at the different features.The way we approached the feature exploration was very simple. First we divided the features into three parts, namely categorical, binary and numerical.



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

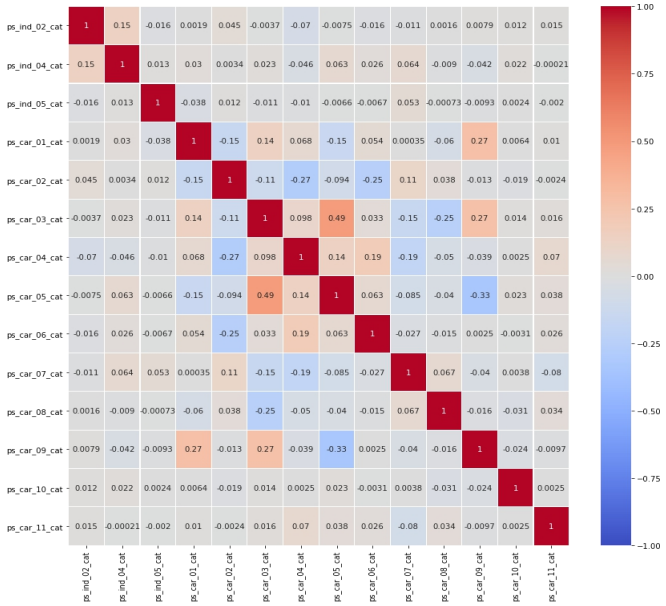
After this, we approached each of the three parts separately. First, we plotted the heat plots for each type of feature to see the correlations. After this, we separately plotted each feature to see the distributions

9.1 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.

9.2 Categorical columns:

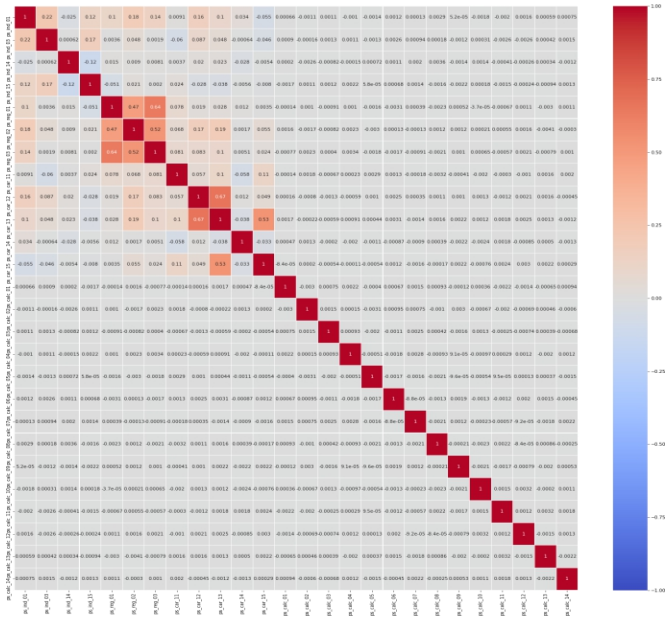


As in the case of binary columns, there wasn't any significant correlation, so we proceeded further.

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

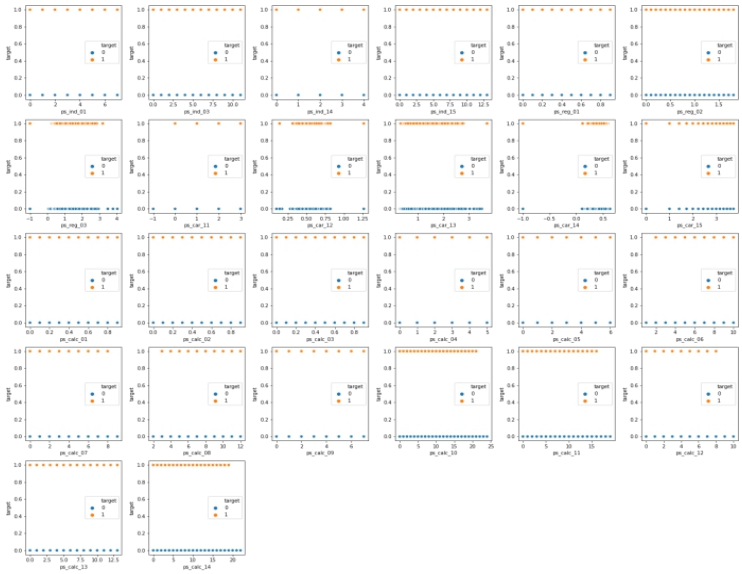
- From these plots, we saw that except a few features (that had "calc" in their column names), most of them were dominated by a single value.
- Some features like 'ps_ind_05_cat' and 'ps_car_04_cat' mostly consists of a single value. Therefore the mode of these features can be used for filling their 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.

9.3 Numerical columns:



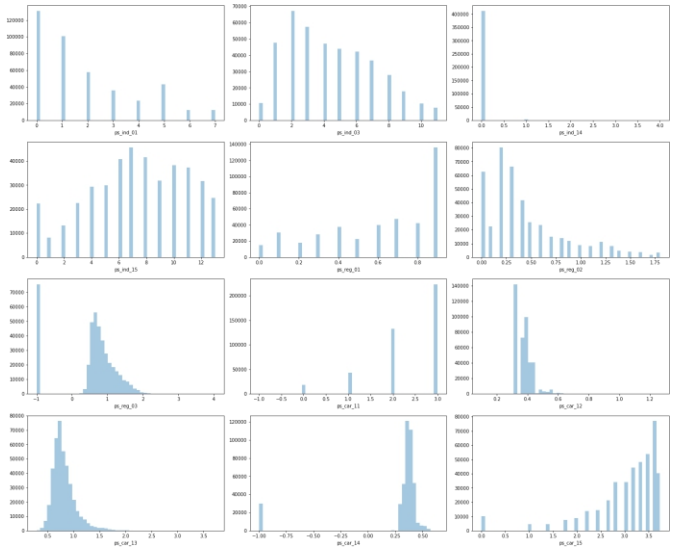
In the numerical columns especially, we found some columns had the word “calc” in their names. Though we did think that this word might stand for “calculated”, we initially didn’t pay enough attention and moved on. Also, since the explanation of each feature has not been disclosed by the company, there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

Hence, we decided to first check the data by scatter plots.



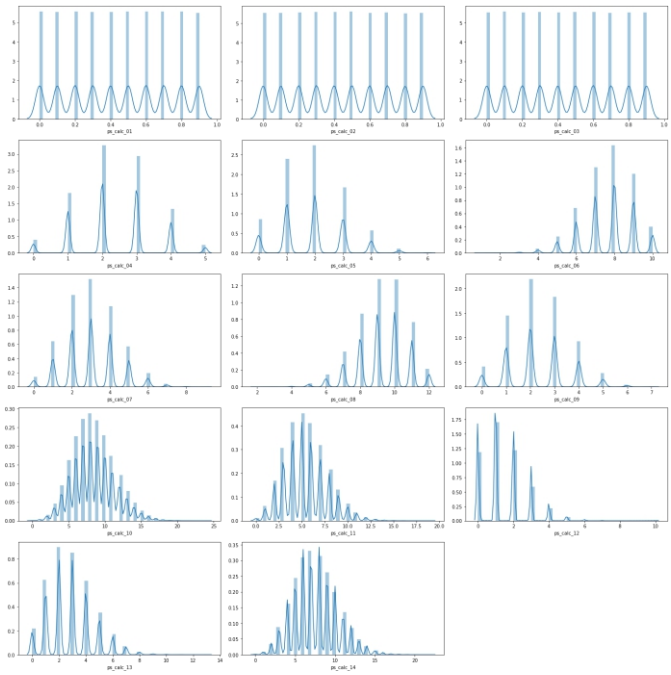
After this, we were still not convinced. So we divided the numerical columns into further two parts, one comprising of “calc” features, and the other without them.

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.

10.ALGORITHMS:

10.1 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

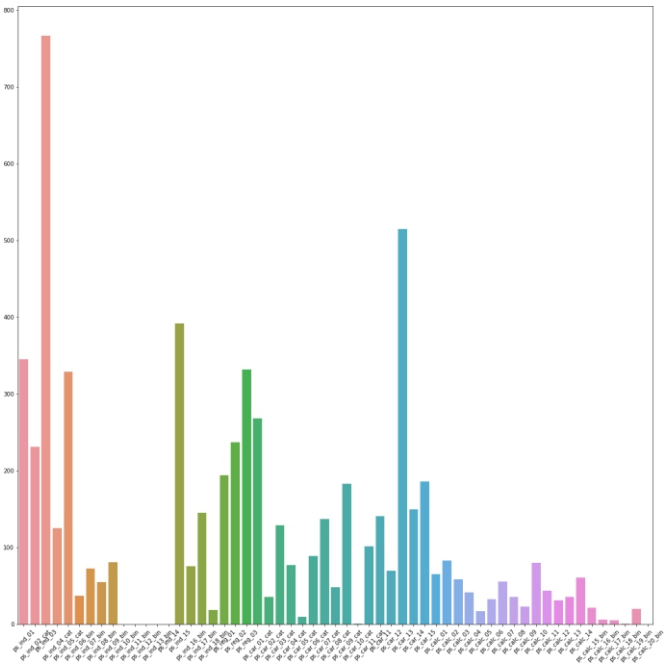
→ Number of leaves

→ The sample training fraction

We used RandomizedSearchCV from sklearn'smodel_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.

10.2 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

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_ca r_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 (d ue 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.

11.EVALUATION CRITERIA

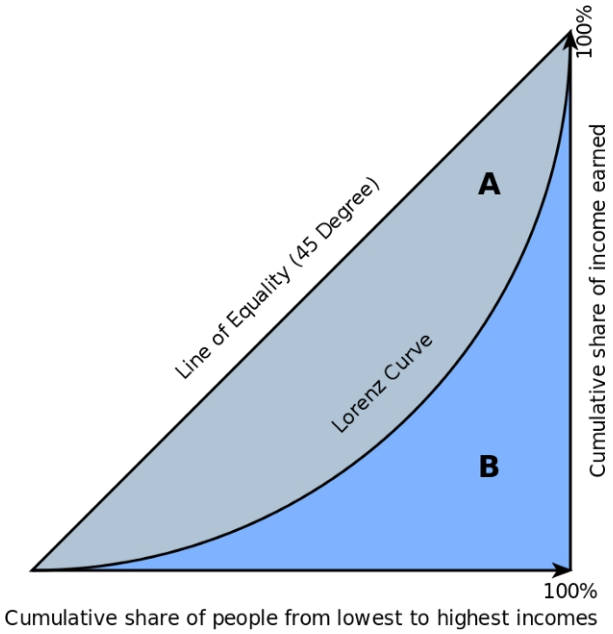
The evaluation criteria of the competition was normalized gini index.

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)



12.TRAINING MODELS

12.1 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.

12.2 Stochastic Gradient descent

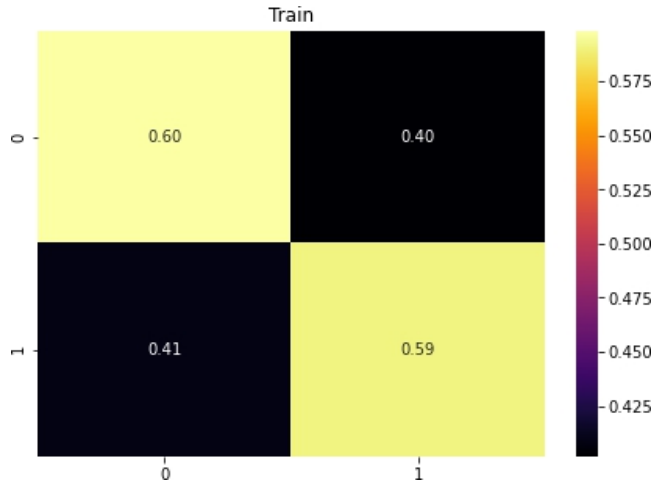
We used the SGDClassifier available in sklearn.

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.

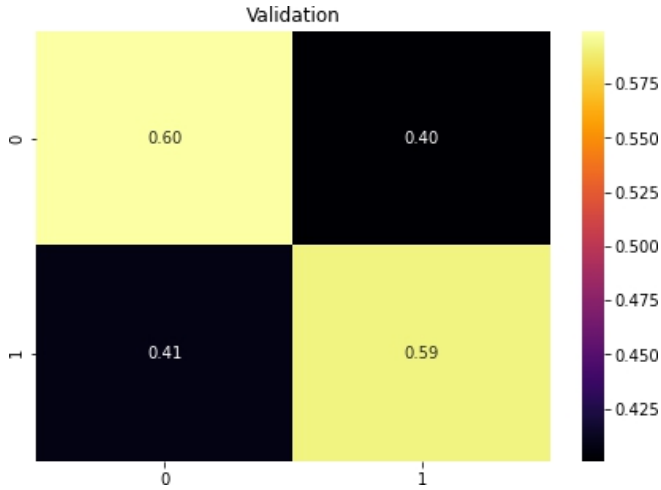
Not much tuning was required. Using the loss function as log loss and class_weight to balanced, we chose the optimum learning rate to achieve a score of about 0.26.

Here is the performance evaluation of SGDClassifier:

- Training data



- Validation data

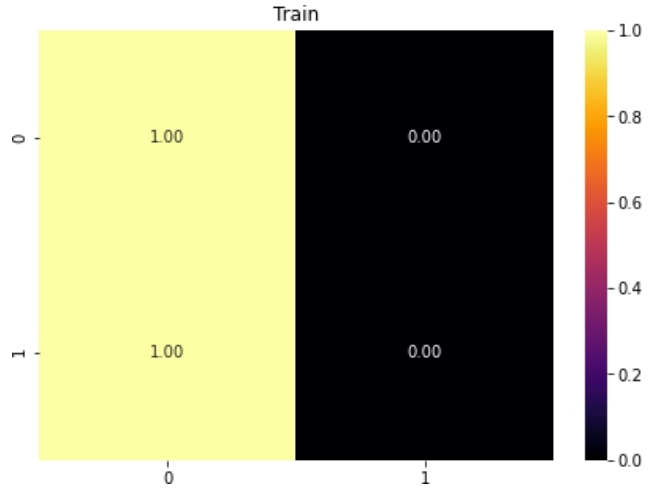


12.3 SVM

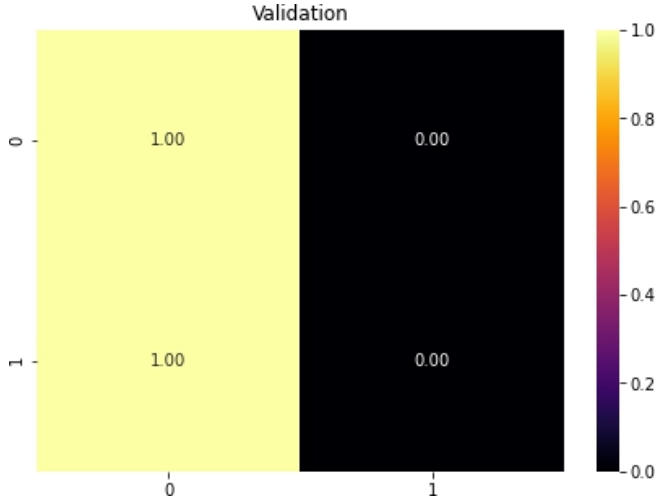
The objective of the support vector machine algorithm is to find an appropriate hyperplane in an N-dimensional space (N, the number of features) that best classifies the data points. SVM uses mathematically defined functions called kernels that take data as input and transform it to compute on it. There are various types of kernels namely linear, non-linear, polynomial, radial basis function (rbf) and sigmoid 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



12.4 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.

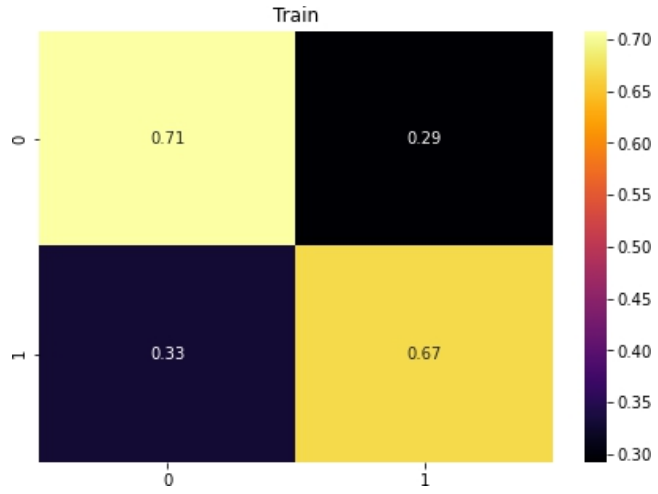
Here are the hyperparameters we tuned along with their final values:

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

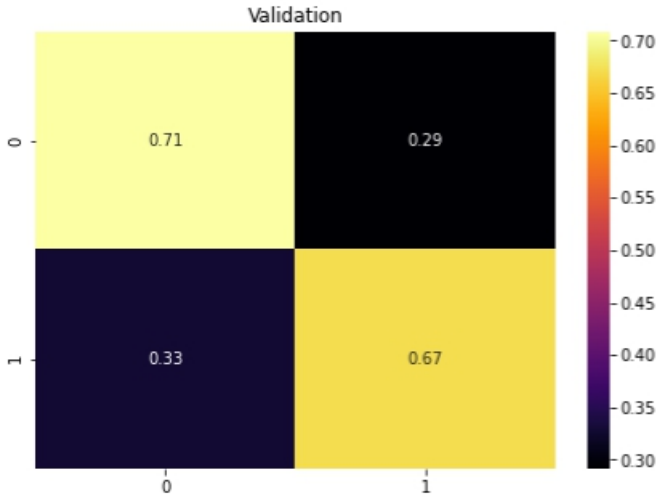
Also, we kept the class_weight to be “balanced” and chose the function to measure the quality of split as gini. Also, the strategy to split was kept as “best”.

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.

12.5 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.

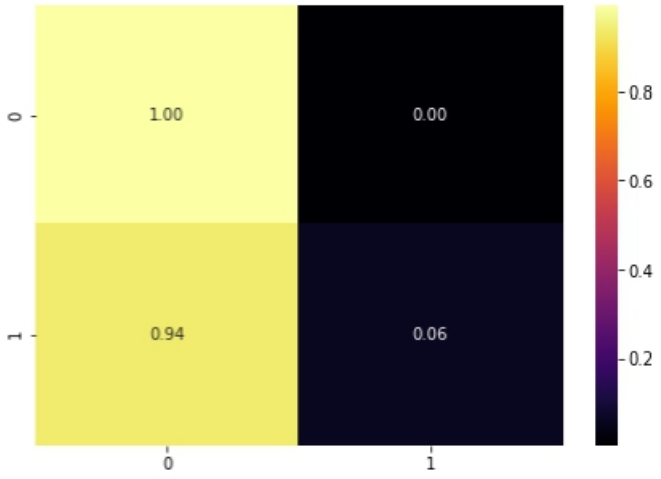
The hyperparameters we tuned were:

Hyperparamter	**Tuned value**
n_estimators	300
depth	9

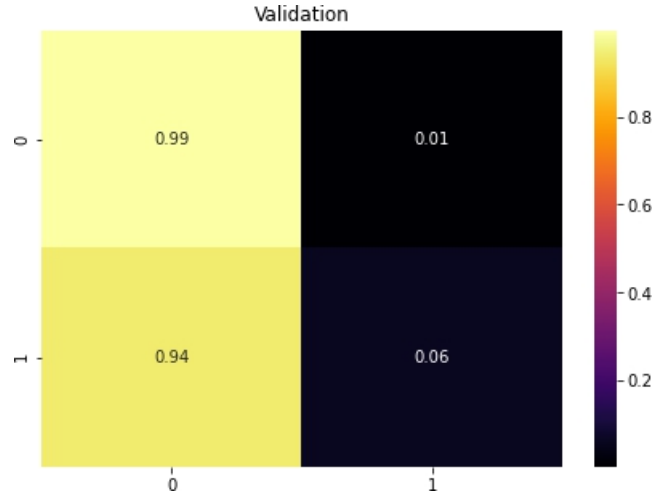
Also, the criterion from split was again kept as “gini” and the out_of_bag score for the model was set to True. This helped in further generalizing the model.

The cross validation gini score obtained was around 0.35 Here is the performance analysis:

- Training data



- Validation data



12.6 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.

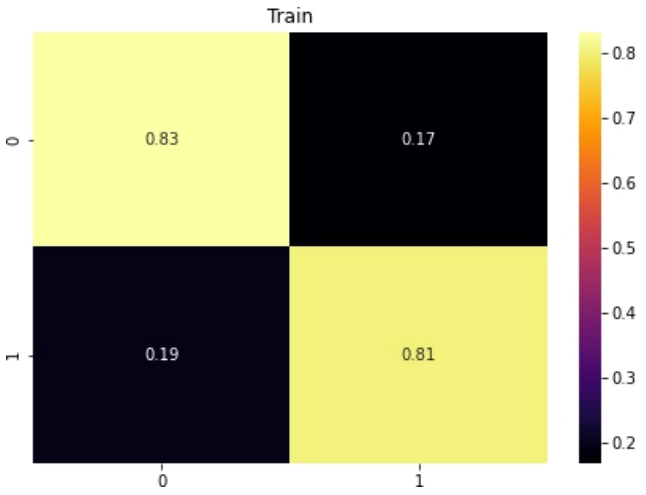
In this, the learners grow sequentially.

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

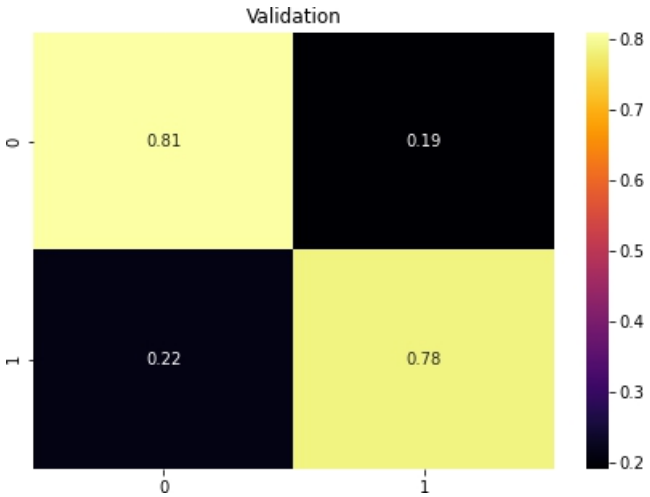
Hyperparameters	**Tuned value**
n_estimators	1000
learning_rate_	0.01

Here is the performance analysis:

- Training data



- Validation data



12.7 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.

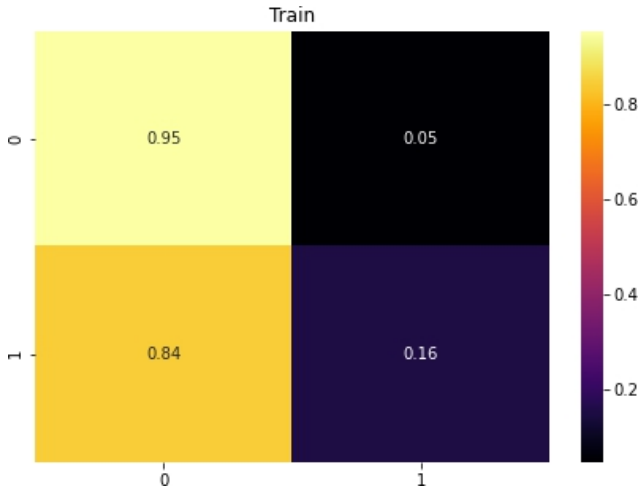
We used randomized search with a 3 fold cross validation in order to tune hyperparameters. But after a few tweaks, we settled in the following:

Hyperparameters	**Tuned value**
n_estimators	250
learning_rate	0.1

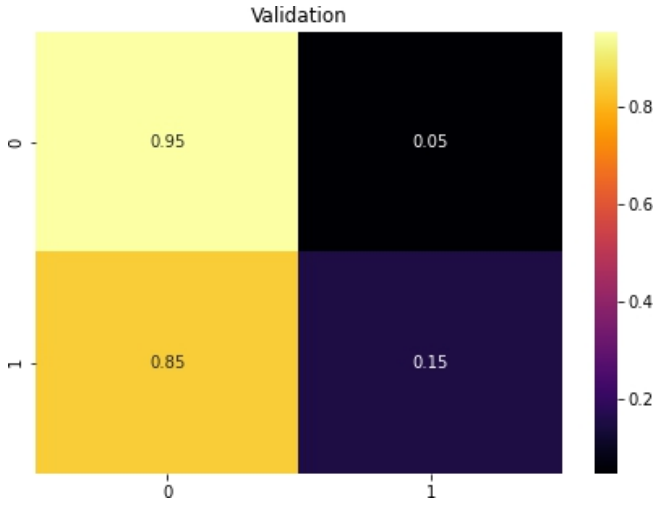
From these, we were able to get a validation score of about 0.33.

Here is the performance analysis:

- Training data



- Validation data



12.8 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.

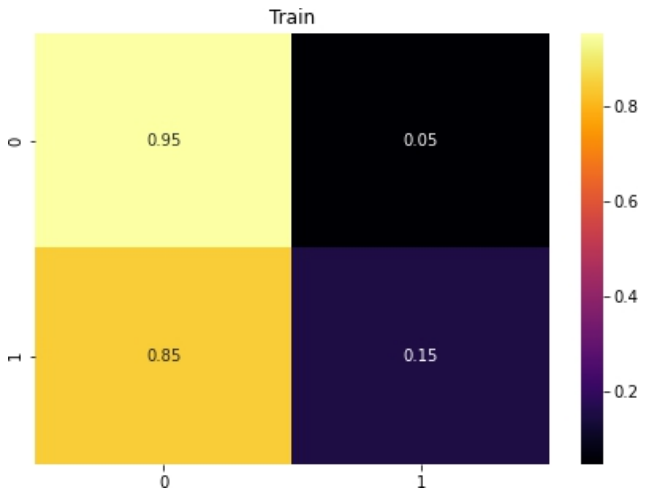
Hyperparameters	Tuned value
n_estimators	200
max_depth	5
learning_rate	0.1
min_child_weight	99
max_leaves	9
reg_lambda	0.5

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.

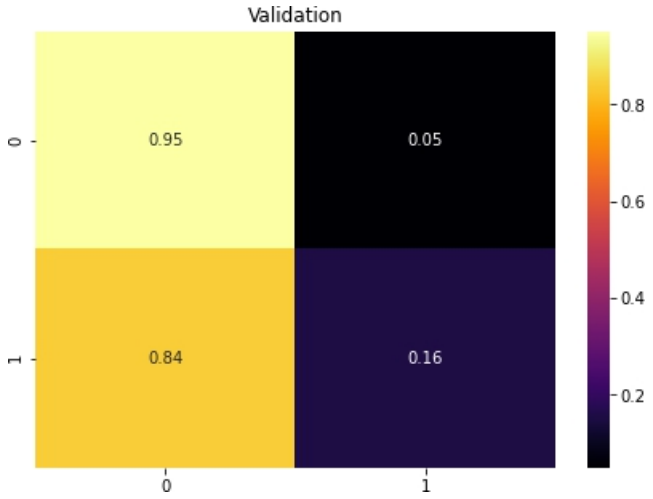
Also, since XGBoost takes time in order to fit on the large datasets, we used the GPU implementation of XGBoost.

We got a validation score of about 0.34. Here are the performance metrics:

- Training data



- Validation data



13 .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.

14.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

15. Code Snippets:

Code for calculating Normalized gini coefficient

```
def gini(actual, pred, cmpcol = 0, sortcol = 1):
    assert( len(actual) == len(pred) )
    all = np.asarray(np.c_[ actual, pred, np.arange(len(actual)) ], dtype=np.float)
```

```
    all = all[ np.lexsort((all[:,2], -1*all[:,1])) ]
    totalLosses = all[:,0].sum()
    giniSum = all[:,0].cumsum().sum() / totalLosses

    giniSum -= (len(actual) + 1) / 2
    return giniSum / len(actual)

def gini_normalized(a, p):
    return gini(a, p) / gini(a, a)

# Impute Missing Values
def impute(df, cols, strat):
    if(strat=="constant"):
        x = input("Enter the string: ")
        imputer= Imputer(strategy=strat, fill_value=str(x))
    else:
        imputer = Imputer(strategy=strat)
    df_impute = df[cols]
    df[cols] = imputer.fit_transform(df_impute)
    return df

## Removing the outliers in the box plots
def remove_outliers(x,df,col):
    q1 = df[col].quantile(q = 0.25)
    q3 = df[col].quantile(q = 0.75)
    iqr = q3 - q1
    outlier_range = 1.5*iqr
    r_whisker = q3 + outlier_range
    l_whisker = q1 - outlier_range
    if (x > r_whisker):
        return q3
    elif (x < l_whisker):
        return q1
    else:
        return x

def check_missing_data(data):
```

```
    d = pd.DataFrame()
    for val in data.columns.values:
        count = data[data[val]==-1].shape[0]
        vals = pd.DataFrame({"Count":count,"%":(count/data[val].shape[0])*100}, index=[val])
        d = d.append(vals)
        #print("% data missing in {} =\t {},\t {:.4f}%".format(val,count,((count/data[val].shape[0])*100)))
    return d

def one_hot_encode(train, test, cat_cols):
    train = pd.get_dummies(train, columns=cat_cols)
    test = pd.get_dummies(test, columns=cat_cols)
    return train, test

def check_data(df):
    vals=df.isnull().sum()
    percent=(100*vals)/len(df)
    type_data=df.dtypes
    diff_vals = df.nunique()
    sk = df.skew()
    table = pd.concat([vals,percent,type_data,diff_vals,sk],axis=1)
    table = table.rename(columns={0:"missing", 1:"missing %", 2:"dtype", 3:"# unique", 4:"skew"})
    return table

def get_scoring_plots(train_pred, test_pred, train_Y, test_Y):
    f, axes = plt.subplots(1, 2, figsize=(16, 5), sharex=True)
    train_cnf = confusion_matrix(train_Y, np.round(train_pred))
    val_cnf = confusion_matrix(test_Y, np.round(test_pred))

    train_cnf = train_cnf / train_cnf.sum(axis=1)[:, np.newaxis]
    val_cnf = val_cnf / val_cnf.sum(axis=1)[:, np.newaxis]

    train_df_cm = pd.DataFrame(train_cnf, index=[0, 1], columns=[0, 1])
    val_df_cm = pd.DataFrame(val_cnf, index=[0, 1], columns=[0, 1])

    sns.heatmap(train_df_cm, annot=True, fmt='.2f', cmap="inferno", ax=axes[0]).set_title("Train")
    sns.heatmap(val_df_cm, annot=True, fmt='.2f', cmap="inferno", ax=axes[1]).set_title("Validation")
    plt.show()
```

Target Analysis

```
if(eda=='y'):
    plt.figure(figsize=(15,8))
    f = sns.countplot(train_target)
    for p in f.patches:
        f.annotate('{:.2f}%'.format(100*p.get_height()/len(train_target)),
                   (p.get_x() + 0.3, p.get_height() + 10000))
```

Analysing and separating the columns

```
def analyse_columns(df):
    data = []
    for col in df.columns:
        if 'bin' in col:
            col_type = 'binary'
        elif 'cat' in col:
            col_type = 'nominal'
        elif df[col].dtype == np.float64:
            col_type = 'interval'
        elif df[col].dtype == np.int64:
            col_type = 'ordinal'

        data_type = df[col].dtype

        data_dict = {
            'column':col,
            'type':col_type,
            'data':data_type
        }

        data.append(data_dict)
    meta = pd.DataFrame(data,columns=['column', 'type', 'data'])
    return meta
```

LightBGM

```
params = {
    "objective":'binary',
    "boosting_type":"gbdt",
```

```
        "is_unbalance":True,
        "learning_rate":0.01,
        "n_estimators":700,
        "reg_alpha":4,
        "reg_lambda":4,
        "num_leaves":10,
        "subsample":0.8
}
lg = lgbm.LGBMClassifier(**params)



i=1
kf = StratifiedKFold(n_splits=3, shuffle=True, random_state=21)
for train_index, test_index in kf.split(train, train_target):
        x_train, x_test = train.iloc[train_index], train.iloc[test_index]
        y_train, y_test = train_target[train_index], train_target[test_index]
        lg.fit(x_train, y_train)
        ans = lg.predict_proba(x_test)[:,1]
        g = gini_normalized(y_test, ans)
        print("KFold {}: {}".format(i,g))
        i=i+1
```

CatBoost:

```
catboost parameters
parameters = dict(
        n_estimators = [1000, 1500, 2000],
        learning_rate = [0.01, 0.03, 0.05],
        depth = [5, 7, 9]
)

cat = catboost.CatBoostClassifier(verbose=False)
cat_rscv = RandomizedSearchCV(estimator=cat, param_distributions=parameters, verbose=1, cv=3,
return_train_score=True, n_jobs=-1)

cat_rscv.fit(train, train_target)
print('Best Score: ', cat_rscv.best_score_)
print('Best Params: ', cat_rscv.best_params_)
```

Logistic regression

```
lr = [0.0001, 0.0002, 0.0003, 0.0004, 0.0005, 0.001, 0.002, 0.003, 0.004, 0.005]
gini_score = {}
for i in tqdm(lr):
    model = SGDClassifier(loss='log', alpha=i, n_jobs=-1, class_weight='balanced')
    model.fit(train_X, target_X)
    gini_score[i] = gini_normalized(target_CV, model.predict_proba(train_CV)[:,1])
    sgd_lr = max(gini_score, key=gini_score.get)
    print("Highest accuracy = ", gini_score[sgd_lr])
    print("Best parameter = ", sgd_lr)
```

SVM

```
svm = SVC(max_iter=100, C=0.5, probability=True, class_weight='balanced', random_state=21)
svm.fit(train_X, target_X)

train_val = svm.predict_proba(train_X)[:,1]
test_val = svm.predict_proba(train_CV)[:,1]

g = gini_normalized(target_CV, svm.predict_proba(train_CV)[:,1])
print(g)
svm.fit(train_ohe, train_target)
ans_svm = svm.predict_proba(test_ohe)[:,1]
```

Naive Bayes:

```
modelG = GaussianNB(var_smoothing =1e+6)
modelG.fit(train_X, target_X)
modelG.get_params(deep=True)


train_val = modelG.predict_proba(train_X)[:,1]
test_val = modelG.predict_proba(train_CV)[:,1]


scorerg = gini_normalized(target_CV,test_val)
print("Gaussian Naive Bayes ",scorerg)
```

Decision Tree

```
parameters = dict(
    criterion = ["gini"],
    splitter = ["best"],
    class_weight = ["balanced"],
    max_features = [23, 25, 27],
    max_depth = [11, 13, 15],
    min_samples_split = [1, 3, 5],
```

```
    min_samples_leaf = [1, 3, 5],
    max_leaf_nodes = [2000, 4000, 6000],
    min_impurity_split = [0.05, 0.1, 0.3]
)
dt = DecisionTreeClassifier()
dt_rscv = RandomizedSearchCV(estimator=dt, param_distributions=parameters, verbose=1, cv=5,
return_train_score=True, n_jobs=-1)

dt_rscv.fit(train_ohe, train_target)
print('Best Score: ', dt_rscv.best_score_)
print('Best Params: ', dt_rscv.best_params_)
```

16.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

17 .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

18.REFERENCES

1. Xu, Guoqing, et al. "Dynamic modeling of driver control strategy of lane-change behavior and trajectory planning for collision prediction." IEEE Transactions on Intelligent Transportation Systems 13.3 (2012): 1138-1155.

2. Xu, Guoqing, Li Liu, Yongsheng Ou, and Zhangjun Song. "Dynamic modeling of driver control strategy of lane-change behavior and trajectory planning for collision prediction." IEEE Transactions on Intelligent Transportation Systems 13, no. 3 (2012): 1138-1155.

3. Kurt, Arda, et al. "Hybrid-state driver/vehicle modelling, estimation and prediction." 13th International IEEE Conference on Intelligent Transportation Systems. IEEE, 2010.

4. Kang, Byeongkeun, and Yeejin Lee. "High-resolution neural network for driver visual attention prediction." Sensors 20.7 (2020): 2030.

5. Montgomery, Douglas C., Elizabeth A. Peck, and G. Geoffrey Vining. Introduction to linear regression analysis. John Wiley & Sons, 2021.

6. Seko, Atsuto, et al. "Representation of compounds for machine-learning prediction of physical properties." Physical Review B 95.14 (2017): 144110.
7. Rosten, Edward, Reid Porter, and Tom Drummond. "Faster and better: A machine learning approach to corner detection." IEEE transactions on pattern analysis and machine intelligence 32.1 (2008): 105-119