GST Hackathon 2024

Introduction

This is a report of the work done on the GST Hackthon2024. The hackathon provides a anonymized dataset with binary target values. This is a classical classification problem. The aim of the work is to build a model to predict the target with good model parameters and evaluation metrics. Firstly let’s explore and analyze the dataset provided and then experiment the models that are suitable for this exercise.

Dataset Description and Exploration

The dataset has the following important attributes:

```
Data columns (total 24 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   ID        785133 non-null  object
 1   Column0   785124 non-null  float64
 2   Column1   785133 non-null  int64
 3   Column2   785133 non-null  float64
 4   Column3   658830 non-null  float64
 5   Column4   657423 non-null  float64
 6   Column5   617953 non-null  float64
 7   Column6   781283 non-null  float64
 8   Column7   785133 non-null  float64
 9   Column8   781283 non-null  float64
 10  Column9   52996 non-null   float64
 11  Column10  785133 non-null  int64
 12  Column11  785133 non-null  int64
 13  Column12  785133 non-null  int64
 14  Column13  785133 non-null  int64
 15  Column14  419430 non-null  float64
 16  Column15  768677 non-null  float64
 17  Column16  785133 non-null  float64
 18  Column17  785133 non-null  int64
 19  Column18  785133 non-null  float64
 20  Column19  785133 non-null  int64
 21  Column20  785133 non-null  int64
 22  Column21  785133 non-null  int64
 23  target    785133 non-null  int64
dtypes: float64(13), int64(10), object(1)
```

- There are many NULL values in the dataset
- Data is mostly numeric except ID column
- The target value is 0 or 1 with following counts:

```
target
0    707772
1     73510
Name: count, dtype: int64
```

As we can see, the data is highly imbalanced.

Data Preprocessing

Lets start analyzing the dataset in more details and take steps to improve the data to make it suitable for modelling.

Handling missing values in the dataset

The dataset contains lot of missing values as we can see in the table below.

	counts	percent_missing
ID	0	0.000000
Column0	9	0.001146
Column1	0	0.000000
Column2	0	0.000000
Column3	126303	16.086829
Column4	127710	16.266034
Column5	167180	21.293208
Column6	3850	0.490363
Column7	0	0.000000
Column8	3850	0.490363
Column9	732137	93.250061
Column10	0	0.000000
Column11	0	0.000000
Column12	0	0.000000
Column13	0	0.000000
Column14	365703	46.578478
Column15	16456	2.095951
Column16	0	0.000000
Column17	0	0.000000
Column18	0	0.000000
Column19	0	0.000000
Column20	0	0.000000
Column21	0	0.000000
target	0	0.000000

Overall missing values in the data ≈ 8.19 %.

Removing samples(rows) having high missing values

Let's check if there are any rows with most number of missing values in 21 columns.

For this purpose, we will define a column threshold as 7 columns which is approx 30% of values in a row. If 70% of columns have missing values, we will prune the dataset as these samples miss important attributes. This could point to errors/incorrect collection of sample data.

We identified 3846 out of 785133 (0.48%) of the total dataset and removed those rows from the dataset.

Feature(Column) wise data analysis

Lets focus now on feature-wise analysis to gain more insights into the data.

Removing columns having null values more than 40%

We are defining a threshold of 60% of the feature/column values to be present to make any meaningful deductions during the modelling process. Hence by considering columns with high number of null values(>40%), we are removing two columns here - Column9 and Column14. So we are now left with 22 columns in total.

Dealing with categorical values

Let's see if any of the features contain few unique values/categorical values. Typically, these features need to be encoded to avoid any numerical bias during model training.

Checking columns that have only one/two/three values in them i.e, constant columns yield the following result:

```
Column10
0    592416
1    188866
Name: count, dtype: int64
Column11
0    537570
1    243712
Name: count, dtype: int64
Column12
0    496211
1    285071
Name: count, dtype: int64
Column13
0    523557
1    257725
Name: count, dtype: int64
Column16
0.0    780339
1.0       939
2.0         4
Name: count, dtype: int64
Column19
0    766603
1     14679
Name: count, dtype: int64
Column20
0    773940
1      7342
Name: count, dtype: int64
Column21
0    779117
1      2165
Name: count, dtype: int64
target
0    707772
1     73510
Name: count, dtype: int64
```

As we can see Column 10, 11, 12, 13, 19, 20, 21 and target features contain binary values. So, these do not need any encoding and can be left alone.

Column 16 has three values – 0.0, 1.0, 2.0 indicating a categorical value like low, medium, high or anything similar.

Column 16

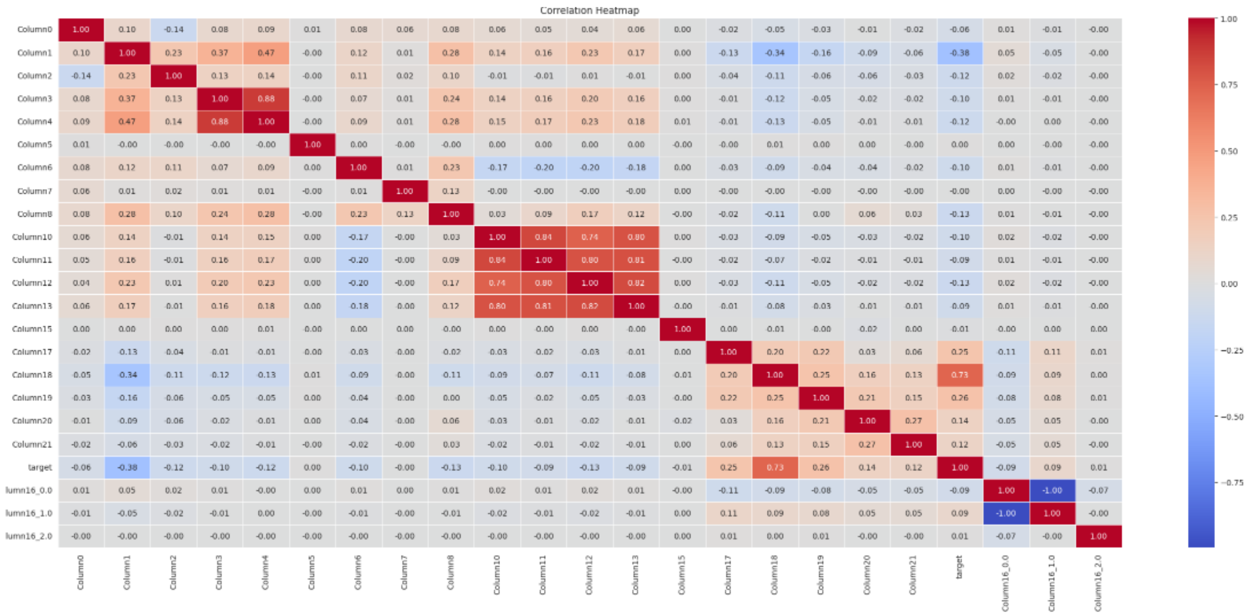
Since the dataset contains only one such feature with three unique values, it makes sense to encode it with one hot encoding mechanism.

Checking for 'Object' features

Apart from those columns, there is an ID column which is an object (non numerical value). This typically doesn't play a role in the model building. Thus, we can safely remove this column.

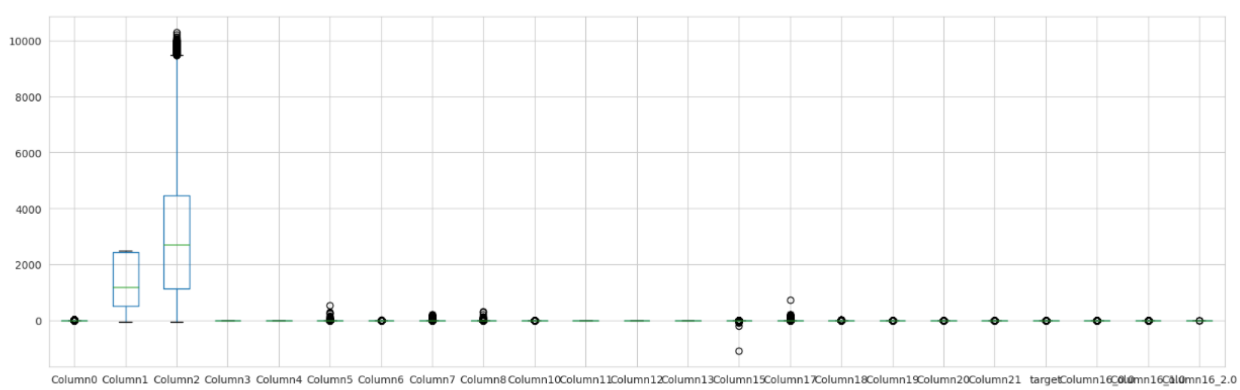
Further data analysis on specific feature - Outliers handling

Lets do further deep-dive into the features and see how the values themselves look like. A correlation map of the given dataset shows below chart:



As you can see Column 10,11,12,13 are highly correlated, but they are categorical values.

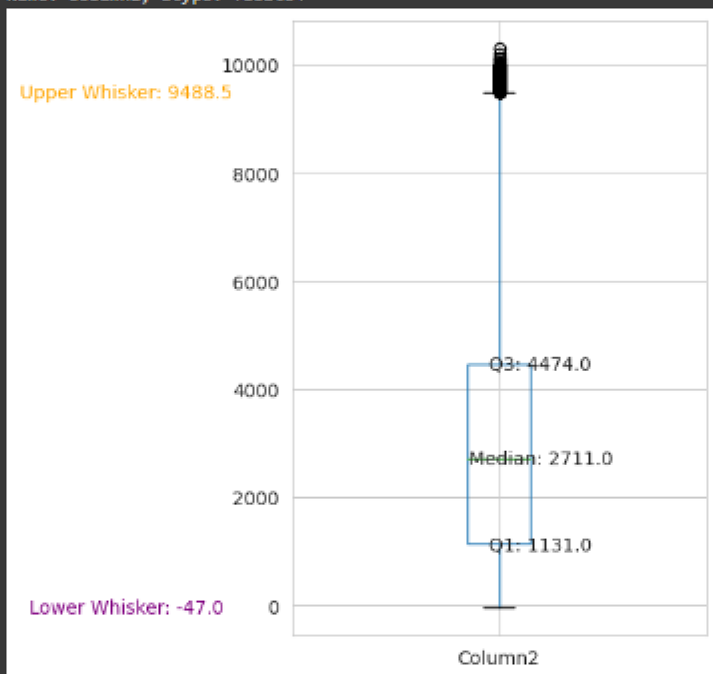
A box plot of the dataset shows how the data is distributed within each feature:



Column2, Column5, Column8, Column15, Column17 have outliers, lets zoom in two those columns.

Column2

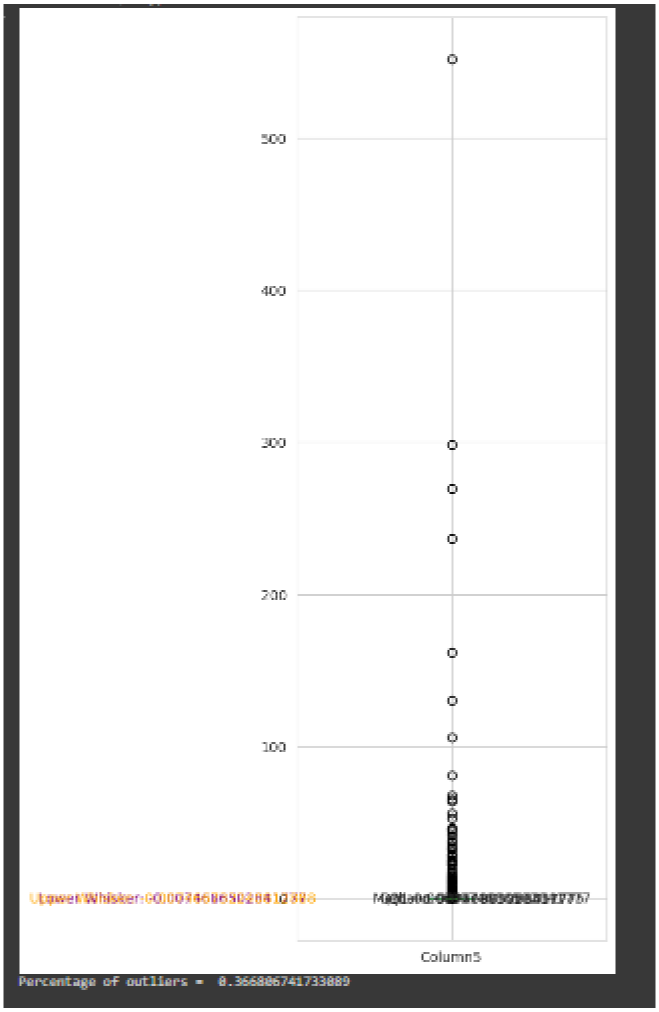
```
count    781282.000000
mean       2952.375478
std        2142.545299
min         -47.000000
25%        1131.000000
50%        2711.000000
75%        4474.000000
max       10290.000000
Name: Column2, dtype: float64
```




```
Percentage of outliers =  0.025598951466947913
```

we can remove these outliers as percentage – 0.025% is very small.

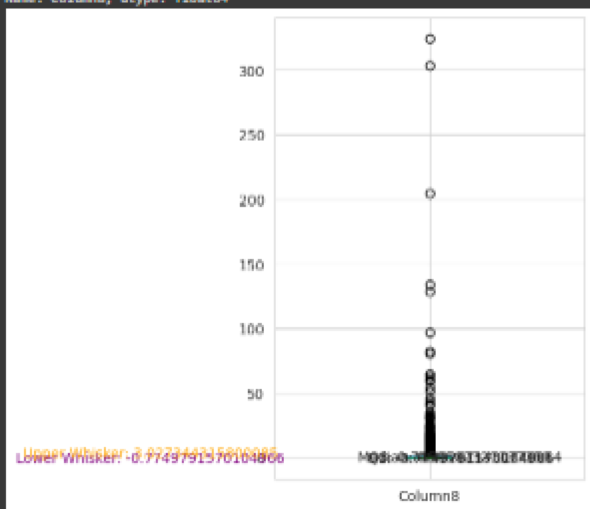
Column5



we can filter out the column5 outliers as the percentage 0.366% is very small.

Column8

```
count    615498.000000
mean          0.196784
std           1.102393
min          -0.774979
25%          -0.774979
50%           0.332671
75%           0.745950
max         323.992484
Name: Column8, dtype: float64
```

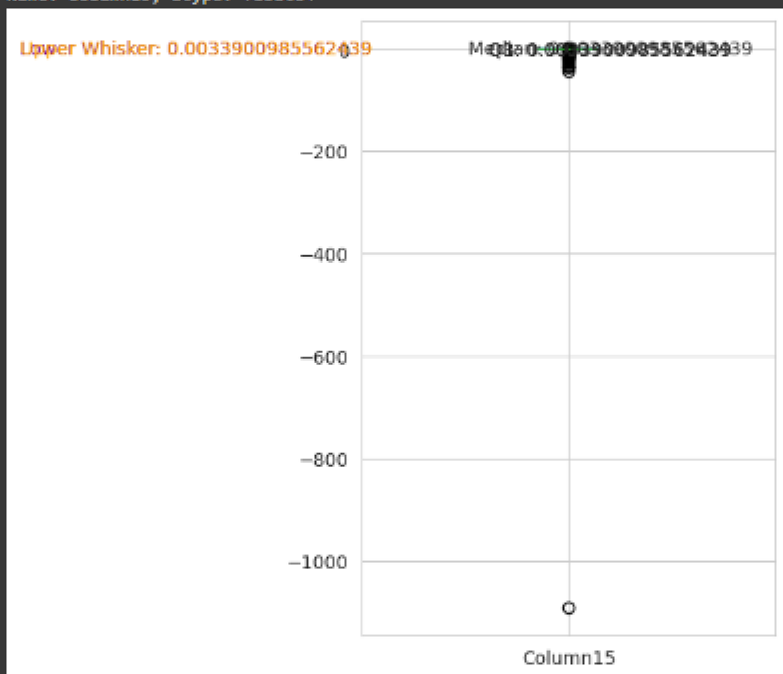



```
Percentage of outliers =  0.16539452605857372
```

We can remove the outliers from column8 as the percentage 0.165% is very small.

Column15

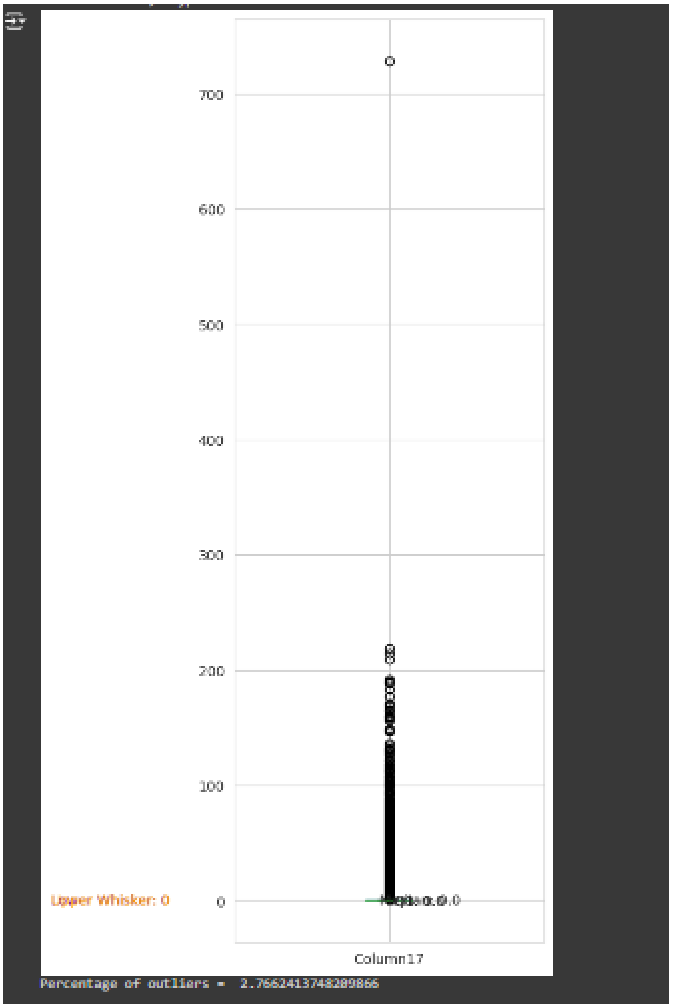
```
count    607123.000000
mean          0.000401
std           1.406641
min       -1091.545904
25%           0.003390
50%           0.003390
75%           0.003390
max           0.012736
Name: Column15, dtype: float64
```




```
Percentage of outliers =  16.240201738362735
```

As the percentage of outliers is 16.24, we will replace and cap the outliers with values of inter quartile ranges – above and below.

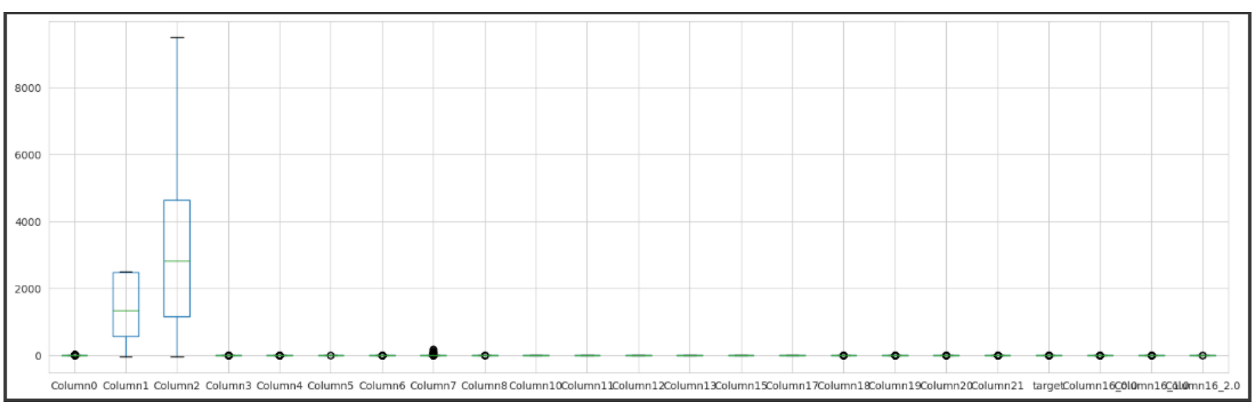
Column17



Column 17 also a high number of outliers – 2.766%, we will cap with values with IQR(Inter Quartile Range) values above and below.

Outliers conclusion

After handling all the outliers, the boxplot looks good as seen below



Handling Missing values in processed dataset

After all the above steps, Let's see still if we need to handle the missing values.

	Percentage of null values
Column15	1.197272
Column3	5.128564
Column4	5.253548

We decide to fill these missing values with median values of those respective features. The same median value is preserved and applied on any predictions on new dataset.

Model Building

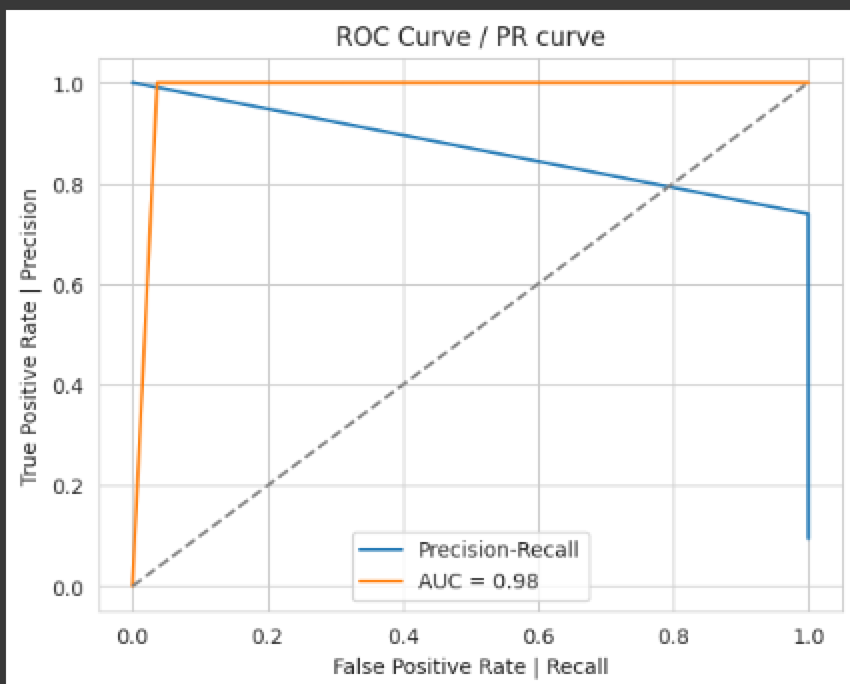
Now that we have a good processed dataset, lets start working on modelling. WE can easily see find that – its a tabular data with binary classification. So we focus on models – Decision Tree, Random Forest, CatBoost, XGBoost, Light GBM and voting classifiers.

Split data into training and testing set

We split the data into 80-20 ratio for training purpose.

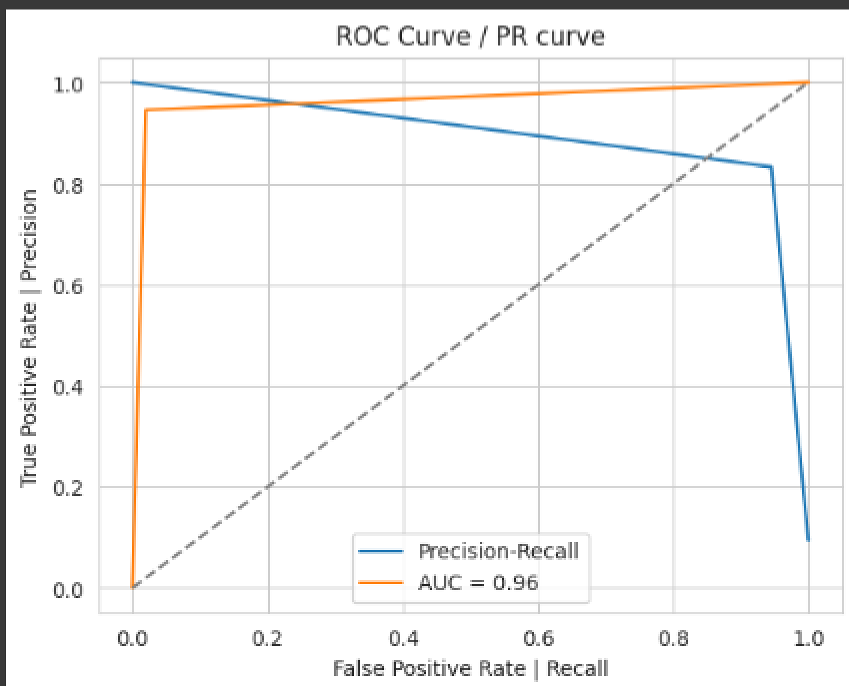
Decision Tree

```
************** Decision Tree ***********************

 Accuracy Score :  0.9667688145747998

Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.96      0.98    237034
           1       0.74      1.00      0.85     24678

    accuracy                           0.97    261712
   macro avg       0.87      0.98      0.92    261712
weighted avg       0.98      0.97      0.97    261712
```



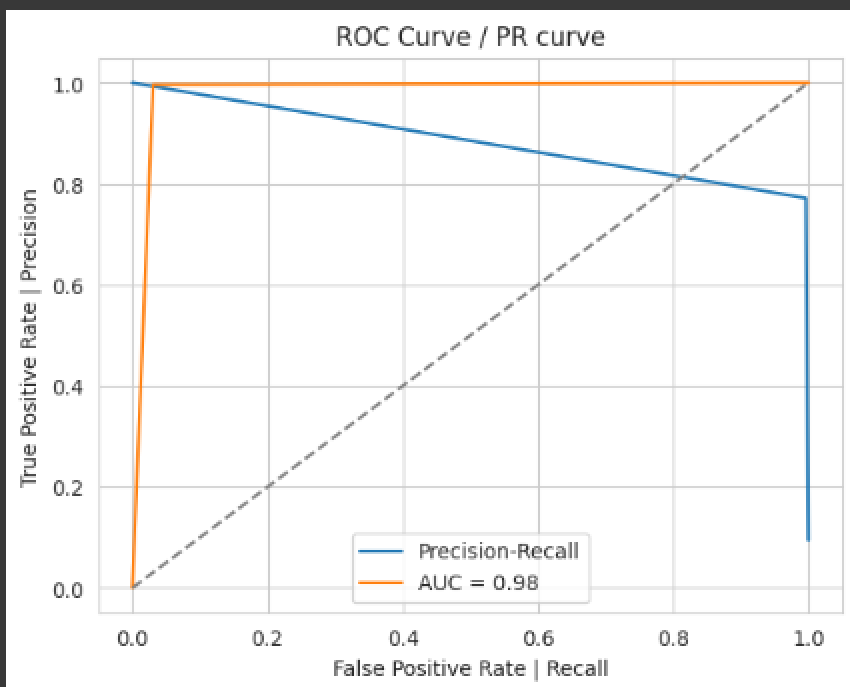
Random Forest

```
************** Random Forest ***********************

 Accuracy Score :  0.9769479427767929

Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    237034
           1       0.83      0.95      0.89     24678

    accuracy                           0.98    261712
   macro avg       0.91      0.96      0.94    261712
weighted avg       0.98      0.98      0.98    261712
```



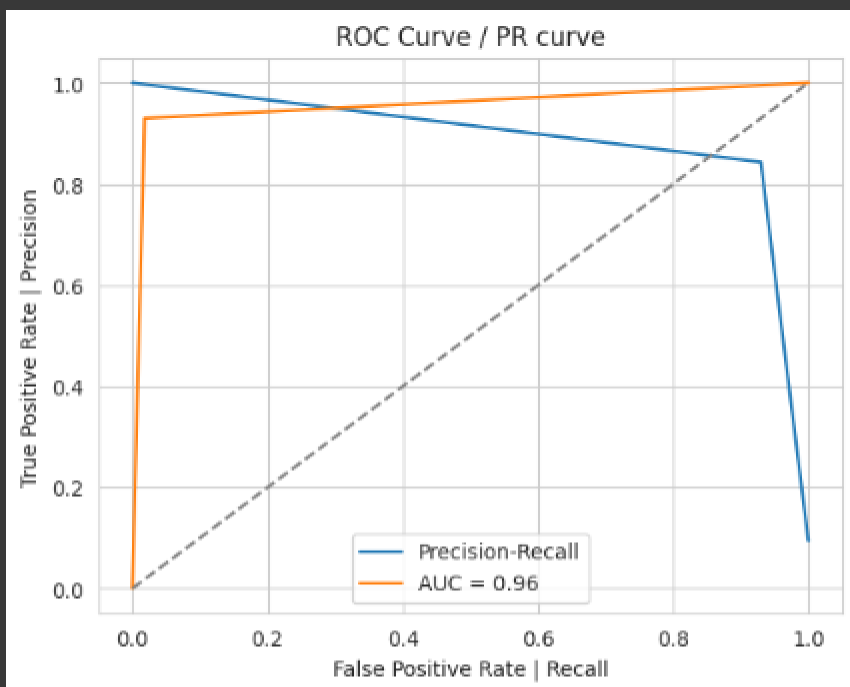
CatBoost

```
************** CatBoost ***********************

 Accuracy Score :  0.9718621996698661

Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.97      0.98    237034
           1       0.77      1.00      0.87     24678

    accuracy                           0.97    261712
   macro avg       0.89      0.98      0.93    261712
weighted avg       0.98      0.97      0.97    261712
```



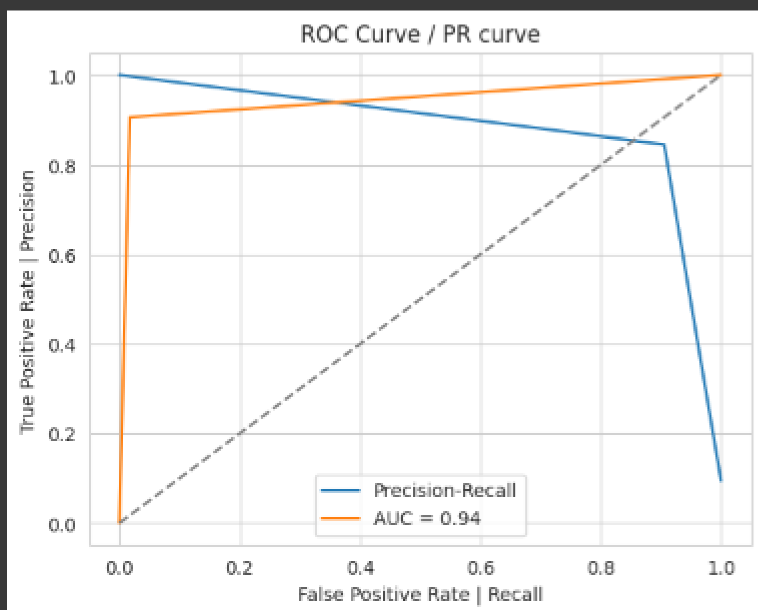
XGBoost

```
************** XgBoost **********************

 Accuracy Score :  0.9770511096166778

Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    237034
           1       0.84      0.93      0.88     24678

    accuracy                           0.98    261712
   macro avg       0.92      0.96      0.94    261712
weighted avg       0.98      0.98      0.98    261712
```



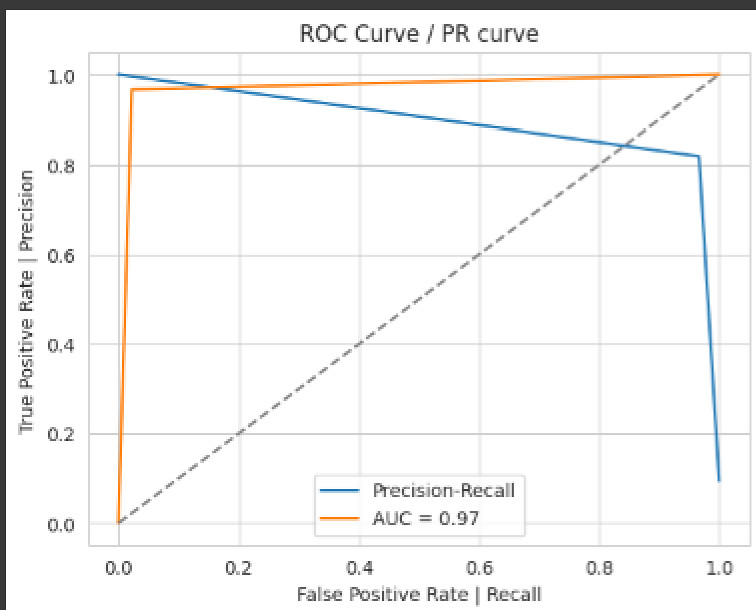
LightGBM

```
************** LGBM **********************

 Accuracy Score :  0.9754883230421226

Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    237034
           1       0.85      0.91      0.87     24678

    accuracy                           0.98    261712
   macro avg       0.92      0.94      0.93    261712
weighted avg       0.98      0.98      0.98    261712
```



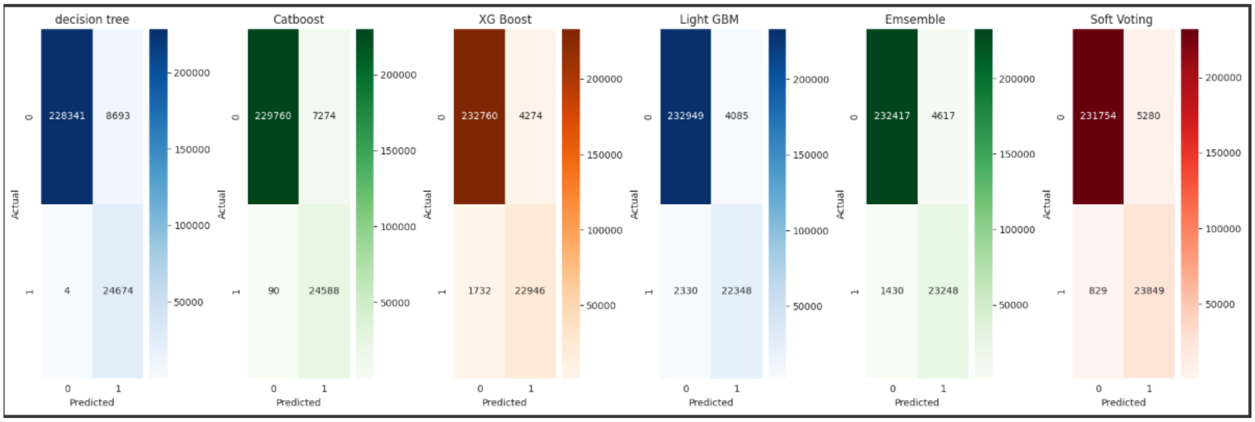
Voting Classifier

```
************** Soft Voting Model ***********************

 Accuracy Score :  0.9766575472274867

Classification Report :
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    237034
           1       0.82      0.97      0.89     24678

    accuracy                           0.98    261712
   macro avg       0.91      0.97      0.94    261712
weighted avg       0.98      0.98      0.98    261712
```

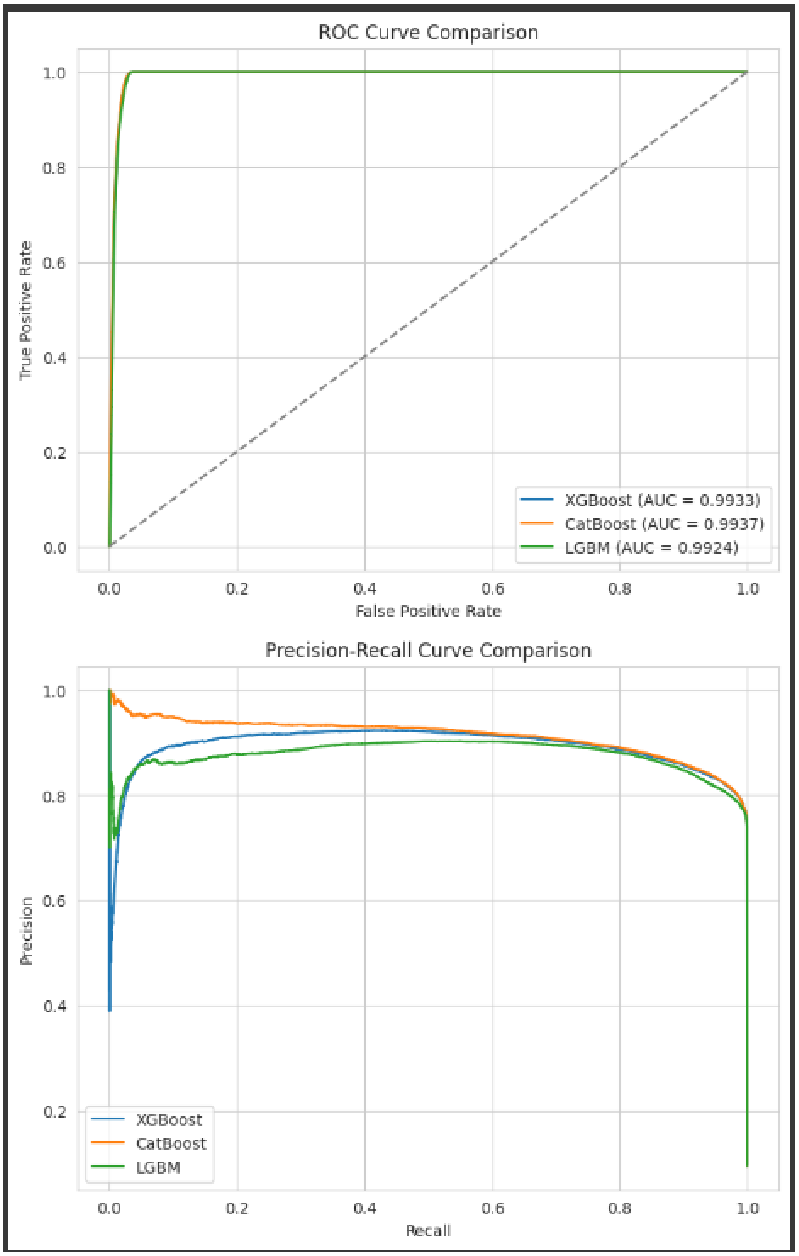


Model Selection

Lets compare each of these model metrics:



The ROC and precision recall curves show below findings:



Conclusion

Decision Tree and Random forest model trials indicate that - tree based models do well for this classification scenarios. While they show good metrics, it can still be improved further by using boosting algorithm based models

Catboost, Xg Boost and LGBM models do very well. Since the data is heavily unbalanced, it makes sense to use a voting classifier based on these 3 boosting models.

It can be seen that Voting classifier does very well, avoids overfitting as expected. It can be further improved by fine tuning underlying models.

Recommended model : Voting Classifier with CatBoost, XG Boost and Light GBM.