TABLE OF CONTENTS:

1. INTROUDUCTION:

Porto Seguro is one of the largest auto and homeowner insurance companies in Brazil. Any inaccuracies in the insurance prediction model of the company can result in a good or cautious driver paying more price or reduction in insurance price for a bad driver. In order to make auto insurance coverage more accessible to more drivers, they handed the dataset to Kaggle to come up with a feasible solution.

2. PROBLEM DESCRIPTION:

In this project we designed and implemented a model that predicts the probability that an auto insurance policy holder will file an insurance claim. This is a Kaggle machine learning competition. We used the provided data set and tried different classifiers to come up with the best classifier among the various ones used.

3. DATASET DESCRIPTION

- The dataset used in this project was taken from Kaggle.(https://www.kaggle.com/c/porto-seguro-safe-driver-prediction) The dataset consists of two csv files - train and test dataset.
- The dataset includes the following features:

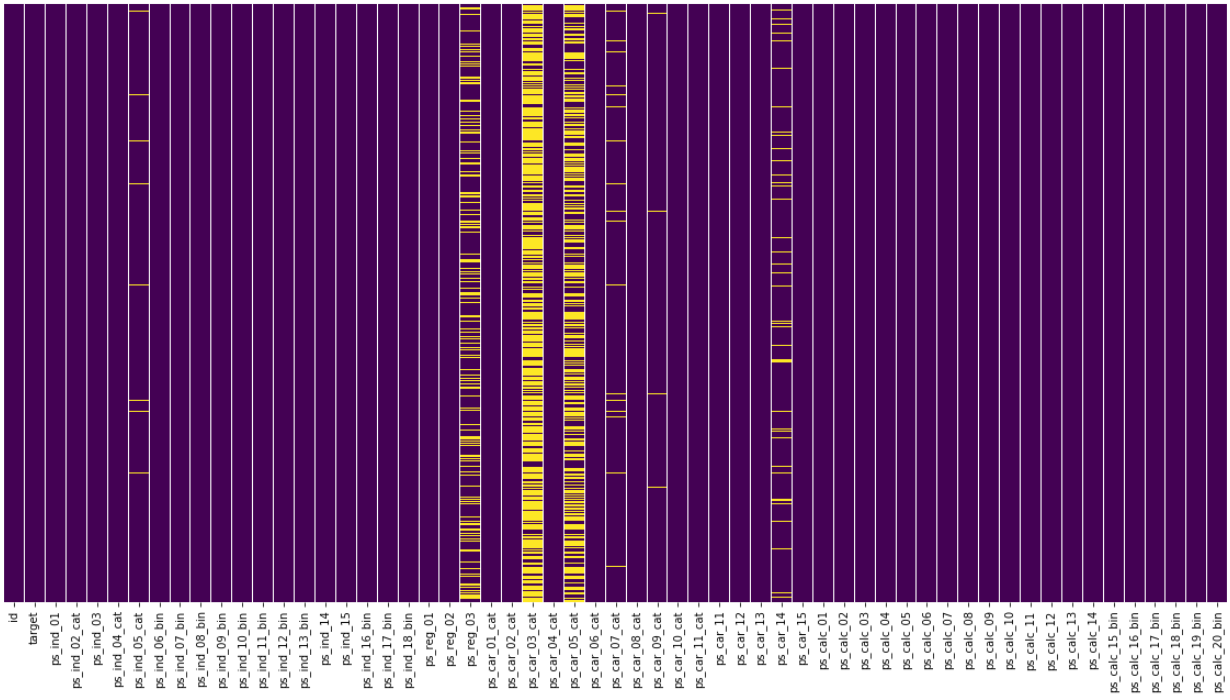
- In each of the dataset, similar groupings features are named as such in feature names(e.g ind,reg,car,calc).
- Feature names also include the postfix “bin” to indicate binary features and “cat” to indicate the categorical features.
- Continuous or Ordinal values are indicated without designations.
- Missing values are indicated as “-1”.
- The resultant value if the claim is filed or not is indicated by the target column in the training dataset.
- As there is no validation dataset, the training dataset is split into training dataset and validation dataset.
- Binary instances are labelled ‘0’ or ‘1’.
- Categorical features integers labelled ‘1’, ‘2’, ‘3’ or ‘4’.
- Other variables are either integer or float values.
- Number of features: 57.
- Output variable is a binary value.
- Number of record in train dataset = 595213
- Number of record in test dataset = 892817

4. PRE-PROCESSING TECHNIQUES

In-order to build a model, we must make sure that the data is properly preprocessed. Since the variable name ending in _cat is an unordered categorical variable and that everything ending in _bin is a binary variable. Everything else is considered to be continuous. We will want to turn the categorical features into factor variables and then perform one-hot encoding.

- Turn the Categorical Features into continuous factor variables.
- One-hot encoding: This method is used to transform features ending with _cat to a format that works with classification and regression algorithms.
- All the features with -1 value-instances are replaced with NA.

Since we should not be performing ML classification algorithms on numeric data which makes it meaningless, we will be performing one-hot encoding. Before we do this however, we need to do something about the missing values before they cause hindrance to our encoding.



4.1 Heatmap reflecting NA/missing values in the dataset across all features.

At the same time, in the given dataset all the missing data is written as -1 rather than the standard “NA”. So, we convert -1’s to NA so as to avoid missing categorical data that doesn’t get encoded as additional factor level.

It is possible that all the features are strongly influential in the dataset because of reasons like the feature might signify only an ID of record, many instances might not be available i.e.; NA, hence to avoid inconsistency in the dataset, a heat-map is used to find the density of missing values in each of the features.

From the above graph, it is observed that the features contain quite a large number of missing values and hence these features are dropped,

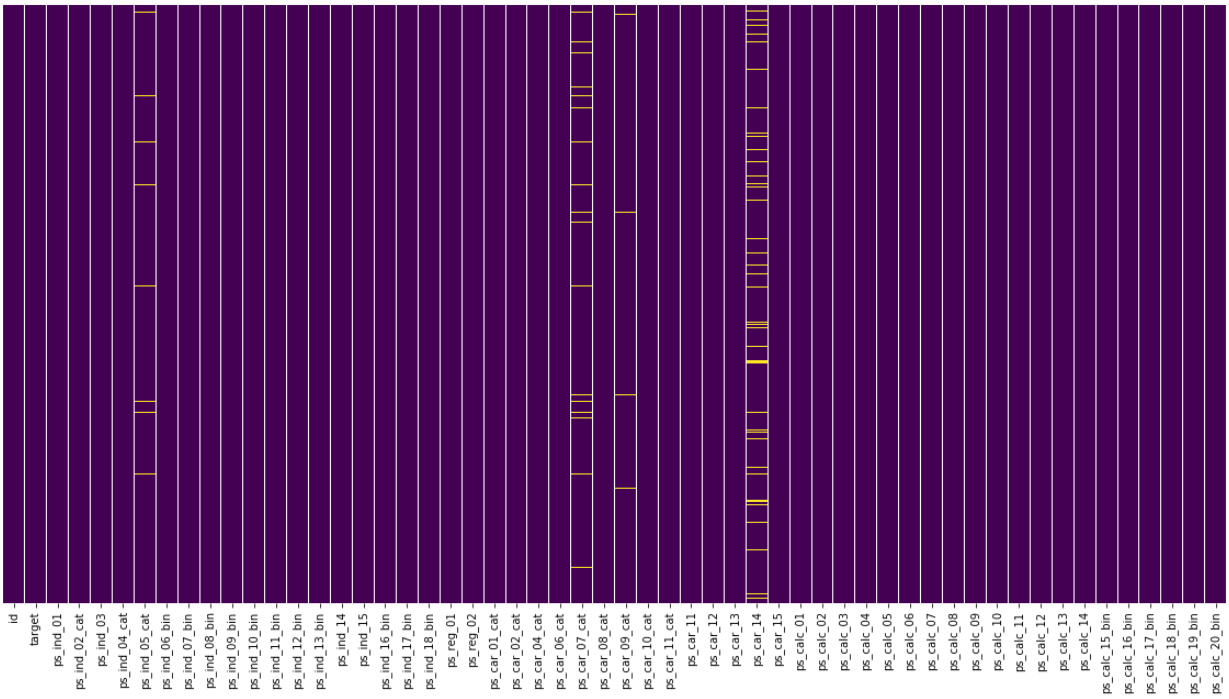
1. ps_reg_03
2. ps_car_03_cat
3. ps_car_03_cat

The above graph i.e.; 4.1 clearly shows the missing values in above stated features.

It is also observed from the heatmap plot that few features have few number of missing value which have to be replaced with a categorical/bin value which is based on following assumptions:

- Consider, that the frequency of type of instances in a feature are almost evenly distributed, then, the mean of the instances is calculated to replace the NA with the mean as shown in 4.3.

- If it is not evenly distributed then the mode of instances is observed, to replace NA with the mode as shown in 4.4.



4.2 Graph reflecting few missing values across all features In dataset

In the graph 4.2,It is observed, there were very few records with all its feature instances as NA, so such records are dropped.

All the defined steps are implemented on test and train data.

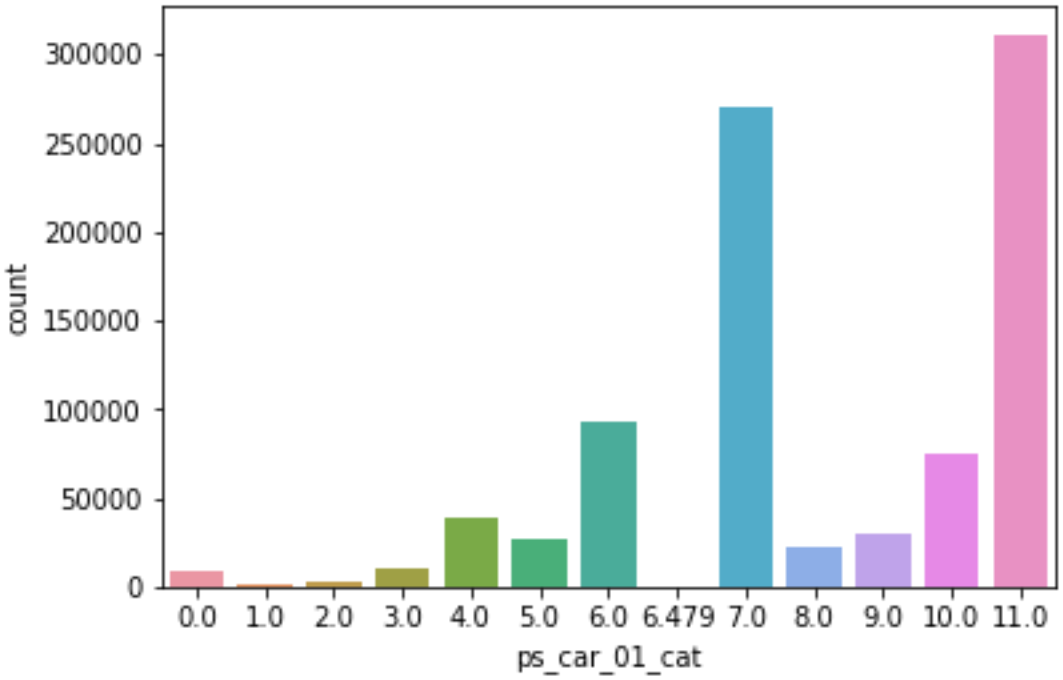


Fig 4.3 Plot showing the count of continuous value of a feature instance in ps_car_01_Cat.

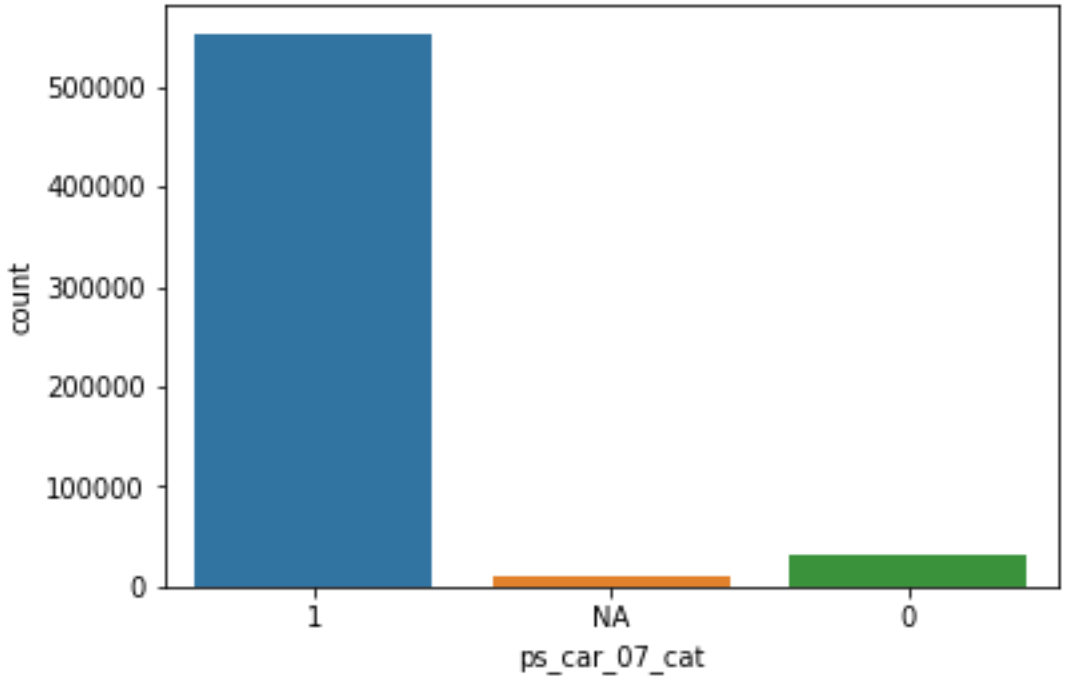


Fig 4.4 Plot showing the count of binary value of a feature instance in ps_car_07_Cat

The overall dataset is quite raw and varies over a large range, the ML algorithms may not work properly without normalization. So, we used the standardized scaler i.e. Standardize features by removing the mean and scaling to unit variance.

On performing all the sequential steps, the datasets seem to have large number of unbalanced data, i.e.; the target variables contain more number of 0s compared to the 1s as shown in Fig 4.5.
Number of 1s: 21694 (Minority Class)

Number of 0s: 573517 (Majority Class)

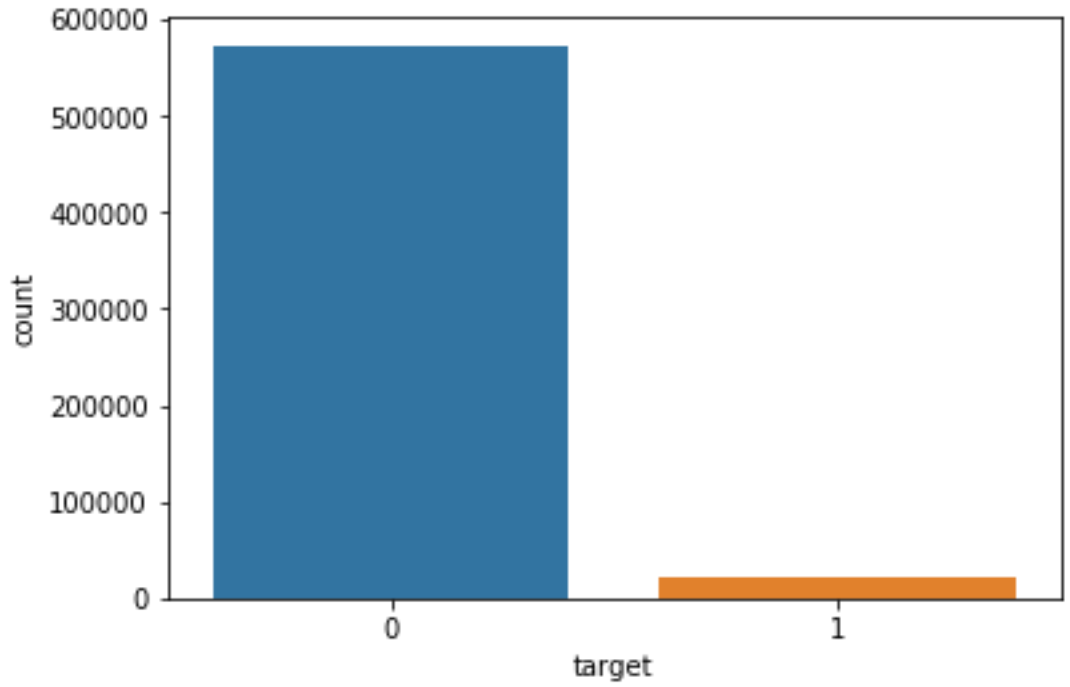


Fig 4.5 Plot stating the uneven feature instance distribution in the dataset.

To maintain the balance between the majority and minority class, we can implement either Up-sampling or Down-sampling that involves randomly duplicating/removing observations from the majority class to reinforce its signal in the algorithm/prevent its signal from dominating the learning algorithm.

Here, we implemented down-sampling because we want to down-sample the Majority class (0) to 100000 from 573517 records and any further Classifier techniques are implemented on this dataset.

Further the data is divided into two splits- Test and Validation in the ratio 2:8 and classifier techniques are implemented.

5. PROPOSED SOLUTION/METHODS:

Various classifiers were used to predict the result of test set by learning from the training dataset. Some of the classifiers used are as follows:

Logistic Regression:

Logistic Regression is an approach to estimate the function mapped from input vector 'X' to output 'Y'. Logistic Regression is a discriminating classifier with linear boundary of separation. Logistic Regression learns the probability of the output given the input.

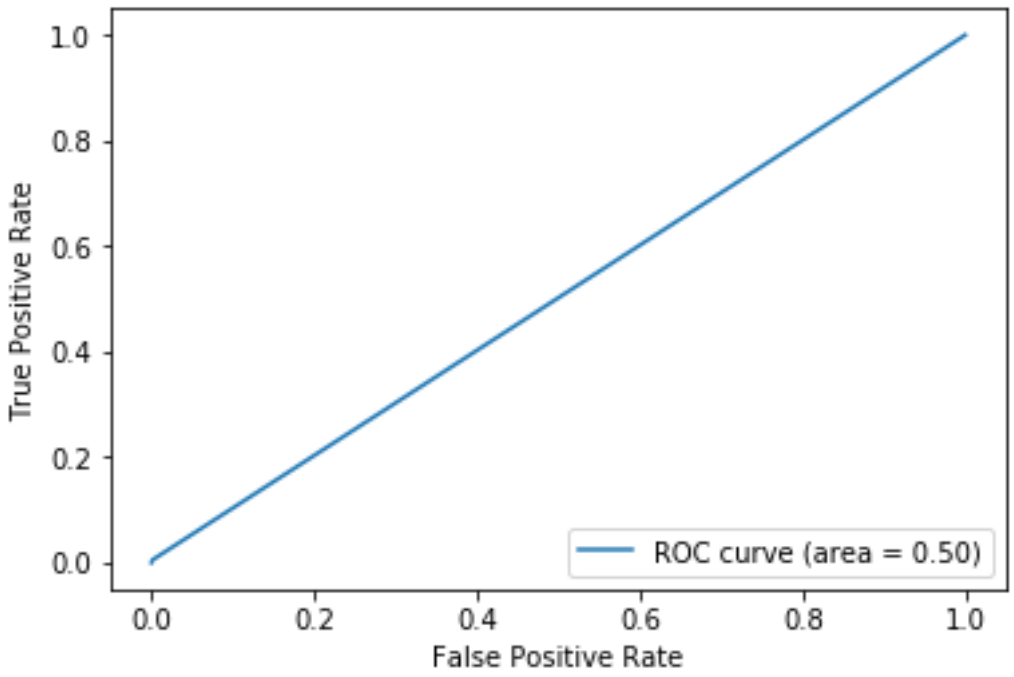
a. Syntax Used:

```
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

b. Parameters Used:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)
```

c. Results:



ROC Curve for Logistic Regression

d. Classification Report and Confusion Matrix:

```
from sklearn.metrics import classification_report,confusion_matrix
```

```
print(classification_report(y_validation,predictions))
```

```
             precision    recall  f1-score   support

          0       0.83      1.00      0.90     20096
          1       0.44      0.01      0.01      4243

avg / total       0.76      0.83      0.75     24339
```

```
print(confusion_matrix(y_validation,predictions))
```

```
[[20067    29]
 [ 4220    23]]
```

Report and Confusion Matrix for Logistic Regression

Decision tree:

Decision Tree is a type of unsupervised learning classifier that learns by makingdecision based on the input instances. It uses decisions rules to build deeper trees for complex algorithm making the model more efficient.

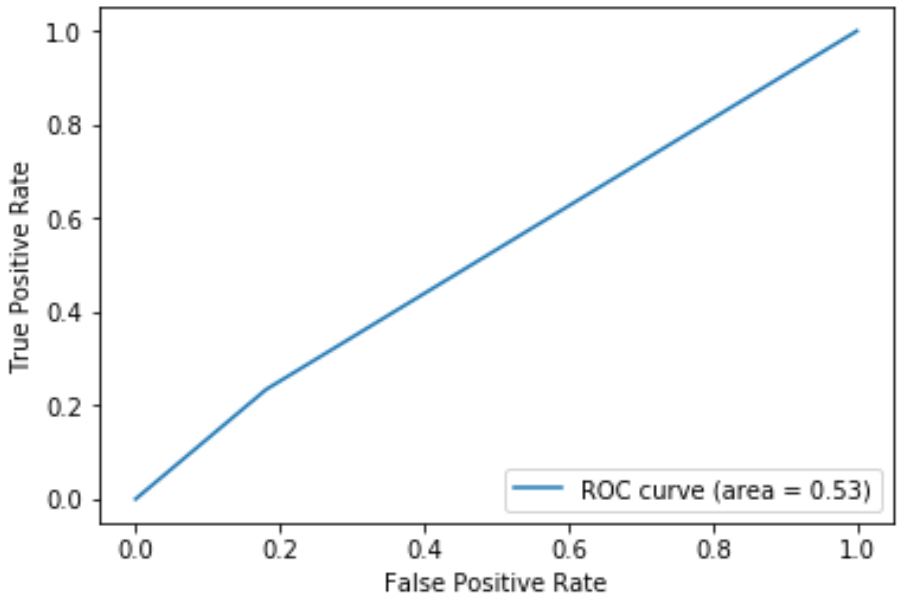
a. Syntax Used:

```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(X_train,y_train)
```

b. Parameters Used:

DecisionTreeClassifier(class_weight=None, criterion='gini',
max_depth=None,max_features=None,
max_leaf_nodes=None,min_impurity_decrease=0.0,
min_impurity_split=None,min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, presort=False, random_state=None, splitter='best')

c. Results:



ROC Curve for Decision Tree

d. Classification Report and Confusion Matrix:

```
pdt = dtree.predict(X_validation)
```

```
print(classification_report(y_validation,pdt))
```

```
             precision    recall  f1-score   support

          0       0.84      0.82      0.83     20096
          1       0.21      0.23      0.22      4243

avg / total       0.73      0.72      0.72     24339
```

```
print(confusion_matrix(y_validation,pdt))
```

```
[[16452  3644]
 [ 3248   995]]
```

Report and Confusion Matrix for Decision Tree

Random Forest Classifier:

Random Forest in an ensemble method of classifier. It is a combination of bootstrap and random attribute selection for each tree model. The training phase includes, creating a bootstrap from the original data sample and creating a decision tree using the attributes. The testing phase takes the aggregated prediction of model based on majority.

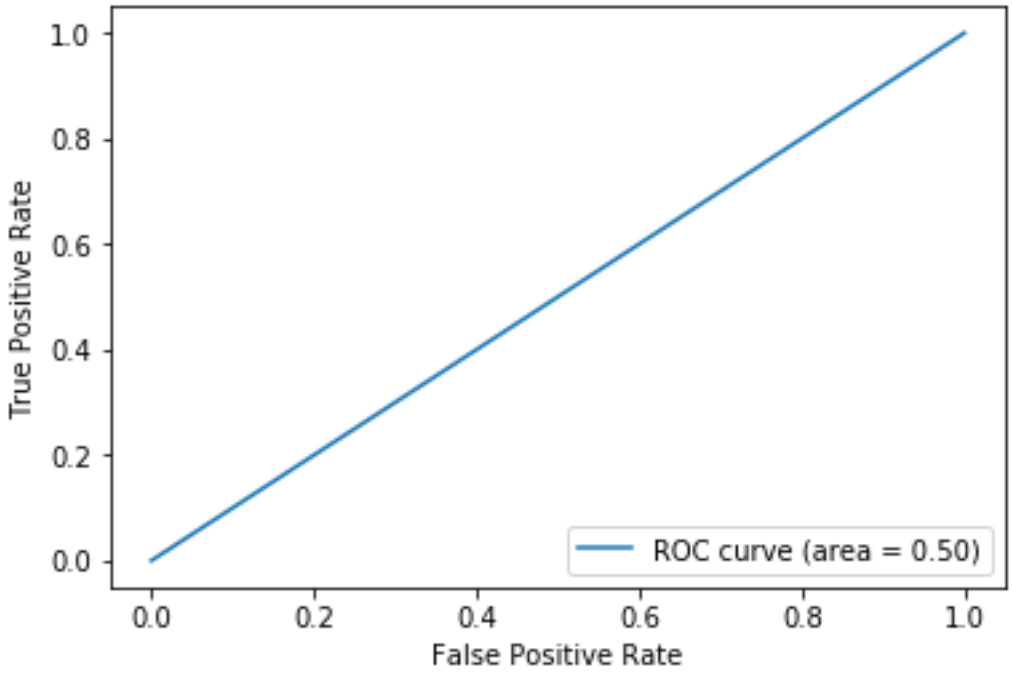
a) Syntax Used:

```
from sklearn.ensemble import RandomForestClassifier
rfc= RandomForestClassifier()
rfc.fit(X_train,y_train)
```

b) Parameters Used:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='sqrt', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=1000, n_jobs=-1,
oob_score=False, random_state=None, verbose=0,
warm_start=False)
```

c) Results:



ROC Curve for Random Forest Classifier

d) Classification report and confusion matrix:

```
print(confusion_matrix(y_validation,rfc_pred))
```

```
[[20096     0]
 [ 4241     2]]
```

```
print(classification_report(y_validation,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.83      1.00      0.90     20096
          1       1.00      0.00      0.00      4243

avg / total       0.86      0.83      0.75     24339
```

Report and Confusion Matrix for Random Forest Classifier.

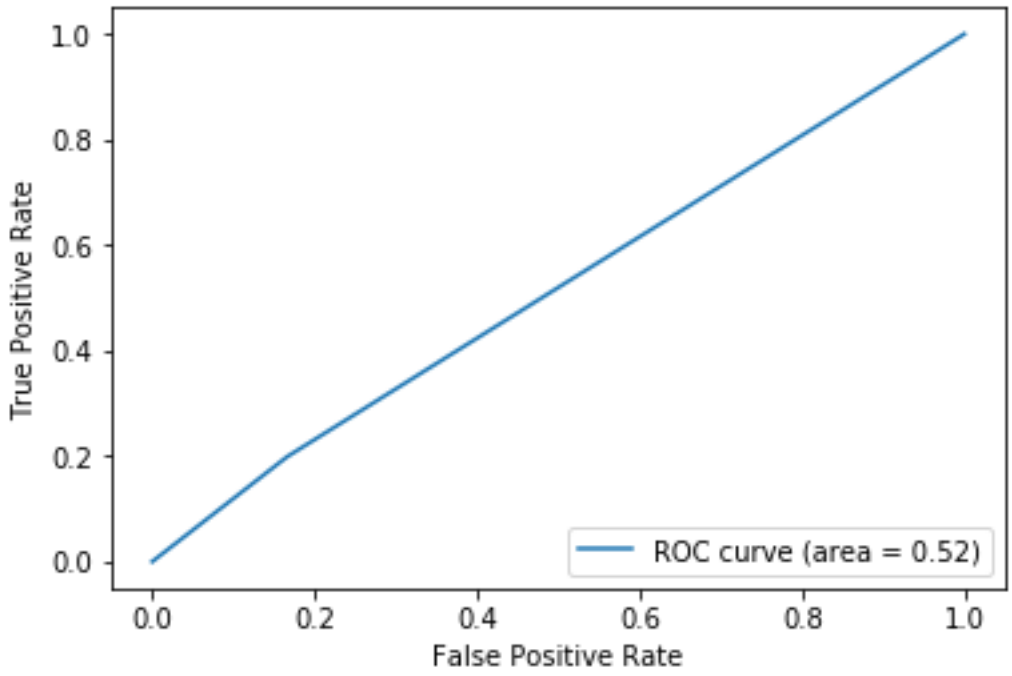
K-Nearest Neighbor:

K- Nearest Neighbor is a type of instance based learning. In the training phase, the input data is stored without any model creation. In the testing phase, when the testing data is loaded to the model, similar data instances are loaded from the memory which is criteria for classification.

a) Syntax:
from sklearn.neighbors import KNeighborsClassifier
Neigh=KNeighborsClassifier(n_neighbours=1)
Neigh.fit(X_train,y_train);

b) Parameters Used:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski', metric_params=None, n_jobs=1, n_neighbors=1, p=2, weights='uniform')

c) Results:



ROC Curve for K-Nearest Neighbor.

d) Classification Report and confusion Matrix:

```
In [141]: print(confusion_matrix(y_validation,pknn))
[[16756  3340]
 [ 3399   844]]

In [142]: print(classification_report(y_validation,pknn))
             precision    recall  f1-score   support

          0       0.83      0.83      0.83     20096
          1       0.20      0.20      0.20      4243

avg / total       0.72      0.72      0.72     24339
```

Report and Confusion Matrix for Random Forest Classifier.

Gradient Boosting:

Gradient boosting is a type of ensemble classifier. it is a combination of Gradient descent and boosting. It is an ensemble classifier of weak prediction models. Usually, the weak prediction models are decision trees.

a. Syntax:

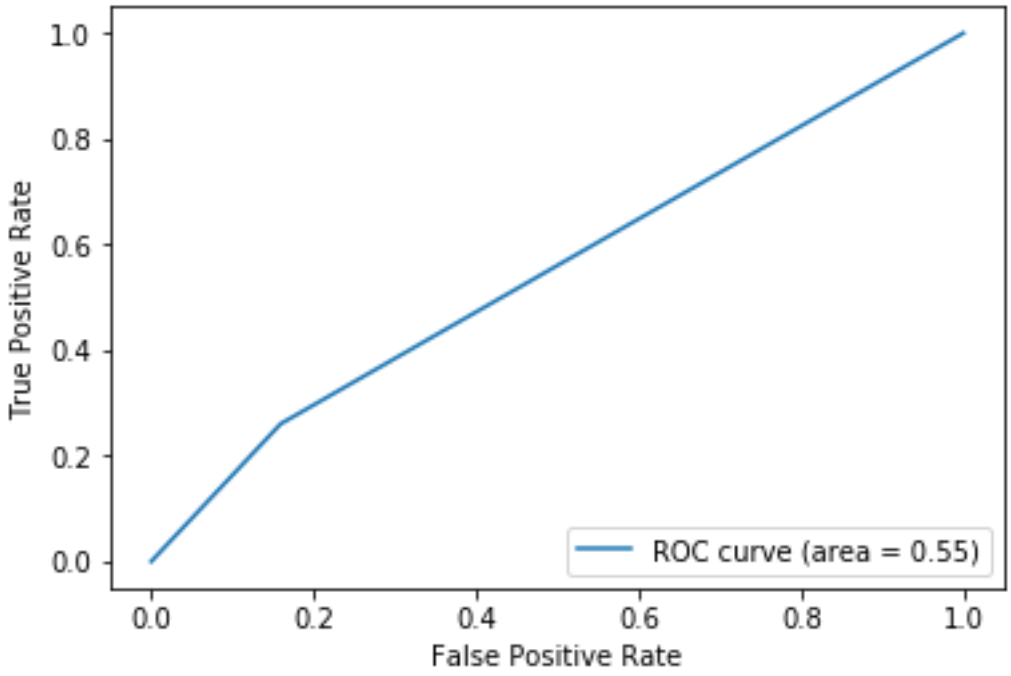
```
from sklearn.ensemble import GradientBoostingClassifier
gb=GradientBoostingClassifier(n_estimators=100,max_depth=5,min_samples_leaf=5,max_features=0.2,random_state=0)
gb.fit(X_train,y_train)
```

b. Parameters Used:

```
GradientBoostingClassifier(criterion='friedman_mse', init=None,learning_rate=0.1, loss='deviance', max_depth=5,
max_features=0.2, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=5, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=101, subsample=1.0, verbose=0,
warm_start=False)
```

c. Results: Also, we use Gini Coefficient. Gini coefficient in Gradient Boosting in used for the predictions

Gini value ranging from 0-100% signifies how strongly the characteristic cannot distinguish good from bad cases.



ROC Curve for Gradient Boosting.

d. Classification Report and confusion Matrix:

```
In [156]: print(confusion_matrix(y_validation,predictionsgb))
          [[16893  3203]
           [ 3138  1105]]

In [157]: print(classification_report(y_validation,predictionsgb))
                       precision    recall  f1-score   support

                    0       0.84      0.84      0.84     20096
                    1       0.26      0.26      0.26      4243

          avg / total       0.74      0.74      0.74     24339
```

Report and Confusion Matrix for Gradient Boosting.

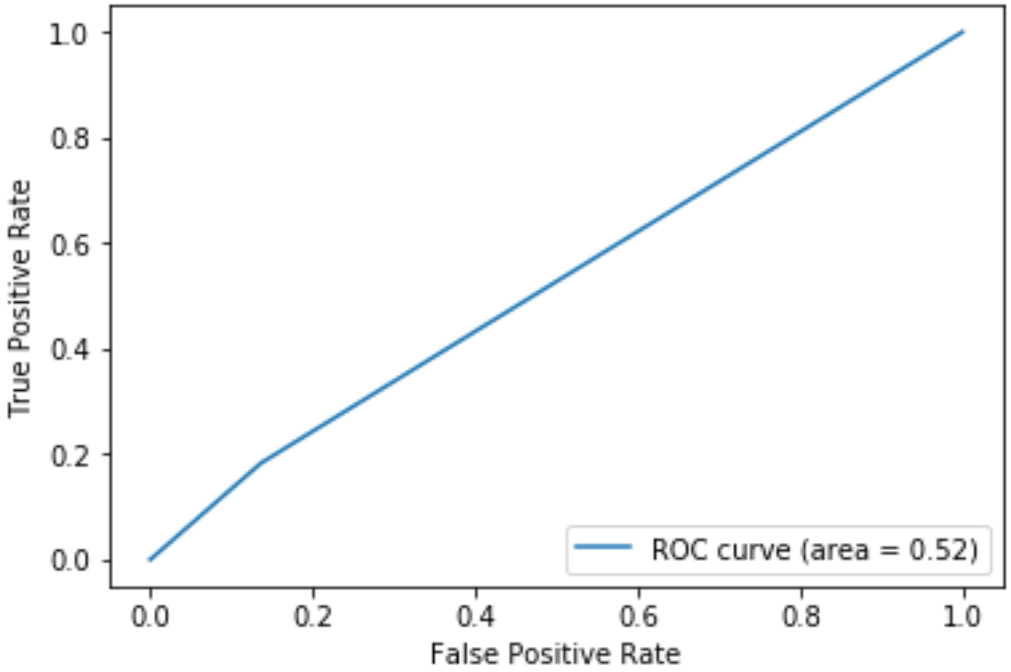
Artificial Neural Network:

Neural Network is a network of perceptrons. It is used to build complex model to obtain minimum training error. The input data is given to the nodes in the intermediate layer known as hidden layer. The output of the last hidden layer is given to the output layer. The activation function used is "relu"(Rectified Linear Unit) as it is quick in evaluation and avoids saturation

a. Syntax:
   clf=MLPClassifier(solver='lbfgs',alpha=1e-5,hidden_layer_sizes=(5,2),random_state=1)
   Clf.fit(X_train,y_train)

b. Parameters Used:
   MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08, hidden_layer_sizes=(500, 1000), learning_rate='constant', learning_rate_init=0.001, max_iter=100000, momentum=0.9, nesterovs_momentum=True, power_t=0.5, random_state=101, shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1, verbose=False, warm_start=False)

c. Results:



ROC Curve for Artificial Neural Network.

d. Classification Report and confusion Matrix:

```
In [163]: print(confusion_matrix(y_validation,neural_netPred))

[[17353  2743]
 [ 3466   777]]

In [164]: print(classification_report(y_validation,neural_netPred))

             precision    recall  f1-score   support

          0       0.83      0.86      0.85     20096
          1       0.22      0.18      0.20      4243

avg / total       0.73      0.74      0.74     24339
```

Report and Confusion Matrix for Artificial Neural Network

6. EXPERIMENTAL RESULTS AND ANALYSIS:

Different Classifiers used:

1. Logistic Regression
2. RandomForestsClassifier

3. K-NN
4. ANN
5. Gradient Boosting
6. Decision Trees

   For the given imbalanced dataset we have implemented and used ROC as our evaluation metric.

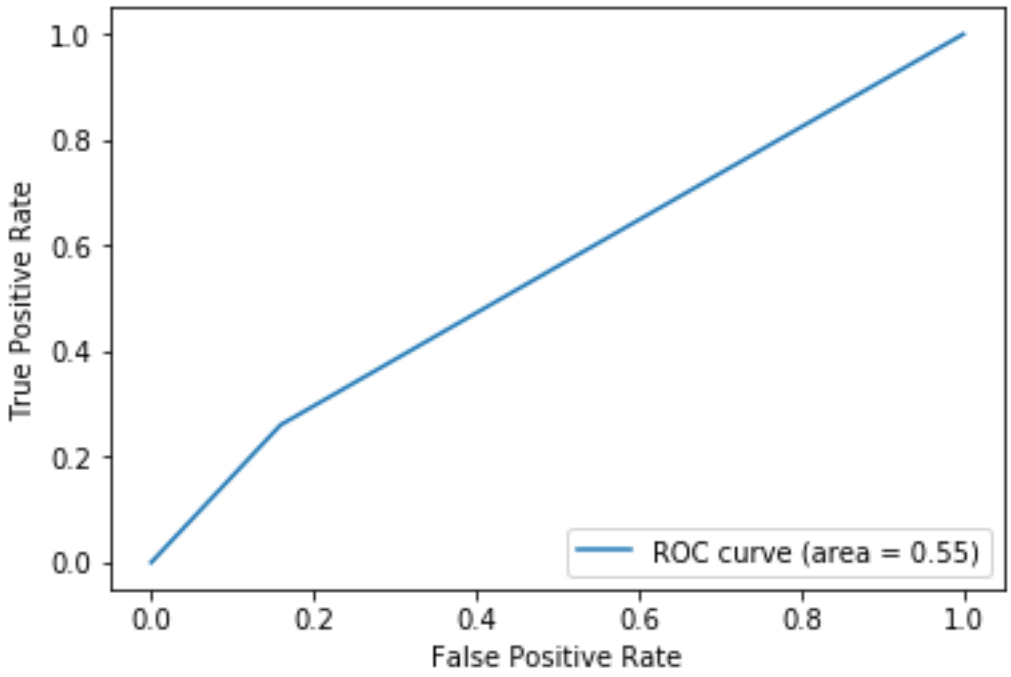
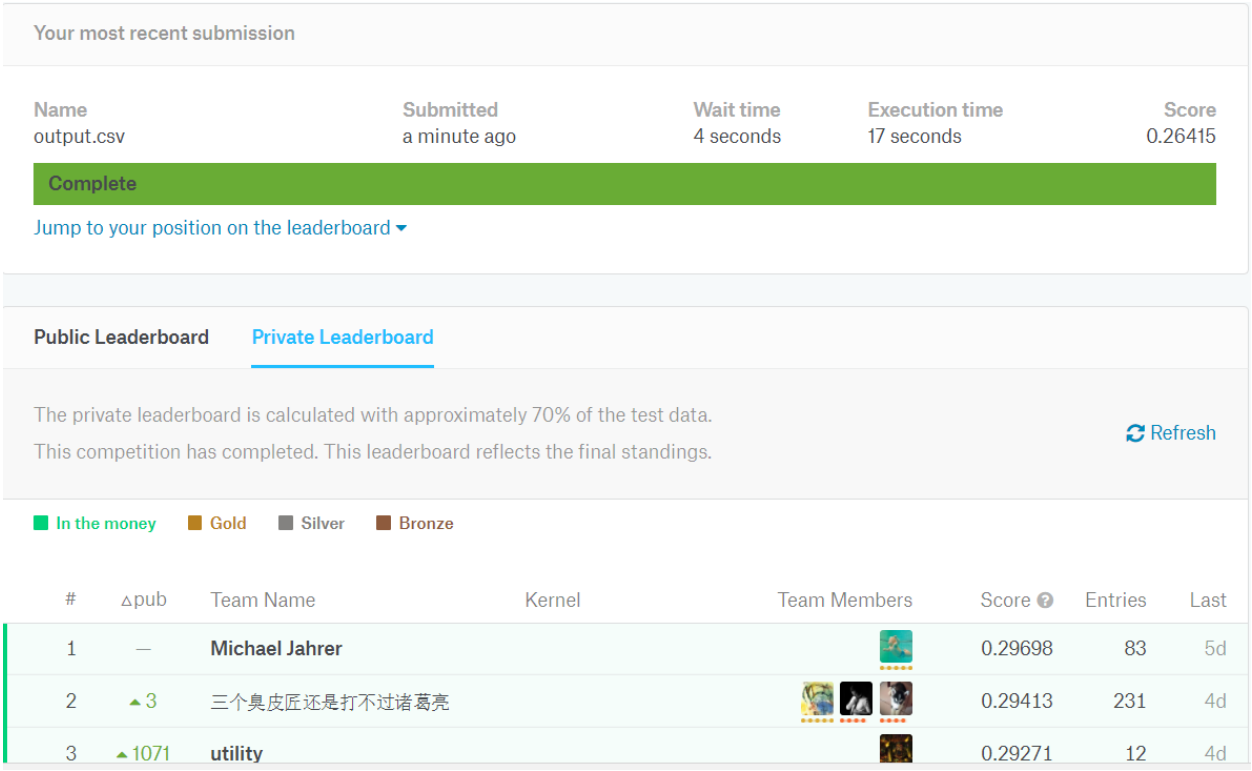


Fig 6.1 ROC curve for the best classifier- Gradient Boosting.

The official competition evaluation metric is gini coefficient for Porto Seguru in Kaggle. It is a statistic measure which measures the ability of a scorecard or a characteristic to rank order risk. The gini coefficient obtained for our project is "**0.26415**"



Screenshot depicting the Gini Coefficient obtained.

7. CONCLUSION:

Among all the classifiers used, Gradient Boosting is the best classifier we have observed with pa rameters:

(criterion='friedman_mse', init=None,
learning_rate=0.1, loss='deviance', max_depth=5,
max_features=0.2, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=5, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100,
presort='auto', random_state=101, subsample=1.0, verbose=0,
warm_start=False)

Since the data is imbalanced accuracy cannot be a valid parameter as an evaluation metric.

8. TOOLS AND LANGUAGES USED:

- Python is used to code the classifiers.
- Scikit Learn for used to import classifiers.
- We used in Anaconda Navigator to launch the Jupyter Notebook to run the code and analyze the data.
- Pandas is used to read from and write into data frames and .csv files.