AIML Project

August 3, 2022

Initially we are importing all the required libraries

[1]:
```
# Importing the libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# Ignore harmless warnings

import warnings
warnings.filterwarnings("ignore")

# Set to display all the columns in dataset

pd.set_option("display.max_columns", None)

# Import psql to run queries

import pandasql as psql
```

[2]:
```
# Load the train data

train = pd.read_csv(r"/home/lab1/Downloads/vehicle_insurance_fraud.csv",␣
    ↪header=0)

# Copy to back-up file

train_bk =train.copy()

# Display first 5 values

train.head()
```

[2]:
```
  Month  WeekOfMonth  DayOfWeek    Make AccidentArea DayOfWeekClaimed  \
0   Dec            5  Wednesday   Honda        Urban          Tuesday
```

```
1   Jan            3  Wednesday   Honda        Urban          Monday
2   Oct            5     Friday   Honda        Urban        Thursday
3   Jun            2   Saturday  Toyota        Rural          Friday
4   Jan            5     Monday   Honda        Urban         Tuesday

  MonthClaimed  WeekOfMonthClaimed     Sex MaritalStatus  Age         Fault  \
0          Jan                   1  Female        Single   21  Policy Holder
1          Jan                   4    Male        Single   34  Policy Holder
2          Nov                   2    Male       Married   47  Policy Holder
3          Jul                   1    Male       Married   65    Third Party
4          Feb                   2  Female        Single   27    Third Party

          PolicyType VehicleCategory  PolicyNumber  RepNumber  Deductible  \
0  Sport - Liability           Sport             1         12         300
1  Sport - Collision           Sport             2         15         400
2  Sport - Collision           Sport             3          7         400
3  Sedan - Liability           Sport             4          4         400
4  Sport - Collision           Sport             5          3         400

   DriverRating PoliceReportFiled WitnessPresent AgentType BasePolicy  \
0             1                No             No  External  Liability
1             4               Yes             No  External  Collision
2             3                No             No  External  Collision
3             2               Yes             No  External  Liability
4             1                No             No  External  Collision

   Fraud_Found
0            0
1            0
2            0
3            0
4            0
```

[3]:

```
# Display the train data information

train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 23 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Month               15420 non-null  object
 1   WeekOfMonth         15420 non-null  int64
 2   DayOfWeek           15420 non-null  object
 3   Make                15420 non-null  object
 4   AccidentArea        15420 non-null  object
```

```
 5   DayOfWeekClaimed    15420 non-null  object
 6   MonthClaimed        15420 non-null  object
 7   WeekOfMonthClaimed  15420 non-null  int64
 8   Sex                 15420 non-null  object
 9   MaritalStatus       15420 non-null  object
 10  Age                 15420 non-null  int64
 11  Fault               15420 non-null  object
 12  PolicyType          15420 non-null  object
 13  VehicleCategory     15420 non-null  object
 14  PolicyNumber        15420 non-null  int64
 15  RepNumber           15420 non-null  int64
 16  Deductible          15420 non-null  int64
 17  DriverRating        15420 non-null  int64
 18  PoliceReportFiled   15420 non-null  object
 19  WitnessPresent      15420 non-null  object
 20  AgentType           15420 non-null  object
 21  BasePolicy          15420 non-null  object
 22  Fraud_Found         15420 non-null  int64
dtypes: int64(8), object(15)
memory usage: 2.7+ MB
```

[4]:
```python
# Count the missing values by each variable, if available

train.isnull().sum()
```

[4]:
```
Month                 0
WeekOfMonth           0
DayOfWeek             0
Make                  0
AccidentArea          0
DayOfWeekClaimed      0
MonthClaimed          0
WeekOfMonthClaimed    0
Sex                   0
MaritalStatus         0
Age                   0
Fault                 0
PolicyType            0
VehicleCategory       0
PolicyNumber          0
RepNumber             0
Deductible            0
DriverRating          0
PoliceReportFiled     0
WitnessPresent        0
AgentType             0
BasePolicy            0
```

```
Fraud_Found          0
dtype: int64
```

By this we can conclude that there are no missing values

[5]:
```
# count no of unique values for each variable

train.nunique()
```

[5]:
```
Month                   12
WeekOfMonth              5
DayOfWeek                7
Make                    19
AccidentArea             2
DayOfWeekClaimed         8
MonthClaimed            13
WeekOfMonthClaimed       5
Sex                      2
MaritalStatus            4
Age                     66
Fault                    2
PolicyType               9
VehicleCategory          3
PolicyNumber         15420
RepNumber               16
Deductible               4
DriverRating             4
PoliceReportFiled        2
WitnessPresent           2
AgentType                2
BasePolicy               3
Fraud_Found              2
dtype: int64
```

[6]:
```
# Drop the variables which are not infulencing on target variable

train =train.drop(['PolicyNumber'], axis=1)

train.head()
```

[6]:
```
  Month  WeekOfMonth  DayOfWeek    Make AccidentArea DayOfWeekClaimed  \
0   Dec            5  Wednesday   Honda        Urban          Tuesday
1   Jan            3  Wednesday   Honda        Urban           Monday
2   Oct            5     Friday   Honda        Urban         Thursday
3   Jun            2   Saturday  Toyota        Rural           Friday
4   Jan            5     Monday   Honda        Urban          Tuesday
```

```
  MonthClaimed  WeekOfMonthClaimed     Sex MaritalStatus  Age          Fault  \
0          Jan                   1  Female        Single   21  Policy Holder
1          Jan                   4    Male        Single   34  Policy Holder
2          Nov                   2    Male       Married   47  Policy Holder
3          Jul                   1    Male       Married   65    Third Party
4          Feb                   2  Female        Single   27    Third Party

          PolicyType VehicleCategory  RepNumber  Deductible  DriverRating  \
0  Sport - Liability           Sport         12         300             1
1  Sport - Collision           Sport         15         400             4
2  Sport - Collision           Sport          7         400             3
3  Sedan - Liability           Sport          4         400             2
4  Sport - Collision           Sport          3         400             1

  PoliceReportFiled WitnessPresent AgentType BasePolicy  Fraud_Found
0                No             No  External  Liability            0
1               Yes             No  External  Collision            0
2                No             No  External  Collision            0
3               Yes             No  External  Liability            0
4                No             No  External  Collision            0
```

[7]:
```python
# display the columns of the dataset

train.columns
```

[7]:
```
Index(['Month', 'WeekOfMonth', 'DayOfWeek', 'Make', 'AccidentArea',
       'DayOfWeekClaimed', 'MonthClaimed', 'WeekOfMonthClaimed', 'Sex',
       'MaritalStatus', 'Age', 'Fault', 'PolicyType', 'VehicleCategory',
       'RepNumber', 'Deductible', 'DriverRating', 'PoliceReportFiled',
       'WitnessPresent', 'AgentType', 'BasePolicy', 'Fraud_Found'],
      dtype='object')
```

[8]:
```python
#Using label encoder

from sklearn.preprocessing import LabelEncoder

LE=LabelEncoder()

# Identify all the object variables

objects=['Month', 'DayOfWeek', 'Make', 'AccidentArea',
       'DayOfWeekClaimed', 'MonthClaimed', 'Sex',
       'MaritalStatus', 'Fault', 'PolicyType', 'VehicleCategory',
       'PoliceReportFiled', 'WitnessPresent', 'AgentType', 'BasePolicy',]

for i in objects:
    train[i]=LE.fit_transform(train[i])
```

```
#check whether objects are encoded or not.
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 22 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Month               15420 non-null  int64
 1   WeekOfMonth         15420 non-null  int64
 2   DayOfWeek           15420 non-null  int64
 3   Make                15420 non-null  int64
 4   AccidentArea        15420 non-null  int64
 5   DayOfWeekClaimed    15420 non-null  int64
 6   MonthClaimed        15420 non-null  int64
 7   WeekOfMonthClaimed  15420 non-null  int64
 8   Sex                 15420 non-null  int64
 9   MaritalStatus       15420 non-null  int64
 10  Age                 15420 non-null  int64
 11  Fault               15420 non-null  int64
 12  PolicyType          15420 non-null  int64
 13  VehicleCategory     15420 non-null  int64
 14  RepNumber           15420 non-null  int64
 15  Deductible          15420 non-null  int64
 16  DriverRating        15420 non-null  int64
 17  PoliceReportFiled   15420 non-null  int64
 18  WitnessPresent      15420 non-null  int64
 19  AgentType           15420 non-null  int64
 20  BasePolicy          15420 non-null  int64
 21  Fraud_Found         15420 non-null  int64
dtypes: int64(22)
memory usage: 2.6 MB
```

By this we can say that all the variables are of numeric type,no variable of object type

[9]:
```
train.head()
```

[9]:
```
   Month  WeekOfMonth  DayOfWeek  Make  AccidentArea  DayOfWeekClaimed  \
0      2            5          6     6             1                 6
1      4            3          6     6             1                 2
2     10            5          0     6             1                 5
3      6            2          2    17             0                 1
4      4            5          1     6             1                 6

   MonthClaimed  WeekOfMonthClaimed  Sex  MaritalStatus  Age  Fault  \
0             5                   1    0              2   21      0
1             5                   4    1              2   34      0
```

```
2            10                   2    1                1    47       0
3             6                   1    1                1    65       1
4             4                   2    0                2    27       1

   PolicyType  VehicleCategory  RepNumber  Deductible  DriverRating  \
0           5                1         12         300             1
1           4                1         15         400             4
2           4                1          7         400             3
3           2                1          4         400             2
4           4                1          3         400             1

   PoliceReportFiled  WitnessPresent  AgentType  BasePolicy  Fraud_Found
0                  0               0          0           2            0
1                  1               0          0           1            0
2                  0               0          0           1            0
3                  1               0          0           2            0
4                  0               0          0           1            0
```

[10]:
```
# Count the target or dependent variable by '0' & '1' and
# their proportion (> 10 : 1, then the dataset is imbalance dataset)

count = train.Fraud_Found.value_counts()
print('Class 0:', count[0])
print('Class 1:', count[1])
print('Proportion:', round(count[0] /count[1], 2), ': 1')
print('Total records', len(train))
```

```
Class 0: 14497
Class 1: 923
Proportion: 15.71 : 1
Total records 15420
```

Here the dataset is imbalance,so we have to do oversampling

[11]:
```
# Identify the independent and Target variables

IndepVar = []
for col in train.columns:
    if col != 'Fraud_Found':
        IndepVar.append(col)

TargetVar = 'Fraud_Found'

x= train[IndepVar]
y= train[TargetVar]
```

[12]:
```python
# Random oversampling can be implemented using the RandomOverSampler class

from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy=0.15)

x_over, y_over = oversample.fit_resample(x, y)

print(x_over.shape)
print(y_over.shape)
```

```
(16671, 21)
(16671,)
```

[13]:
```python
# Splitting the dataset into train and test

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30,␣
 ↪random_state = 42)

#copy for back-up

x_test_bk = x_test.copy()


x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[13]:
```
((10794, 21), (4626, 21), (10794,), (4626,))
```

[14]:
```python
cols1=['RepNumber','Deductible','DriverRating','Month',
       'WeekOfMonth','Make','DayOfWeek','DayOfWeekClaimed',
       'MonthClaimed','WeekOfMonthClaimed','PolicyType']
```

[15]:
```python
# Scaling the features by using MinMaxScaler

from sklearn.preprocessing import MinMaxScaler

mmscaler = MinMaxScaler(feature_range=(0, 1))

x_train[cols1] = mmscaler.fit_transform(x_train[cols1])
x_train = pd.DataFrame(x_train)

x_test[cols1] = mmscaler.fit_transform(x_test[cols1])
x_test = pd.DataFrame(x_test)
```

[16]:
```
Results = pd.read_csv(r"/home/lab1/Downloads/Results.csv", header=0)
Results.head()
```

[16]:
```
Empty DataFrame
Columns: [Model Name, True_Positive, False_Negative, False_Positive,
True_Negative, Accuracy, Precision, Recall, F1 Score, Specificity, MCC,
ROC_AUC_Score, Balanced Accuracy]
Index: []
```

[17]:
```
# Build the Calssification models and compare the results

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import GradientBoostingClassifier
import lightgbm as lgb

# Create objects of classification algorithm with default hyper-parameters

ModelLR = LogisticRegression()
ModelDC = DecisionTreeClassifier()
ModelRF = RandomForestClassifier()
ModelET = ExtraTreesClassifier()
ModelKNN = KNeighborsClassifier(n_neighbors=5)
ModelSVM = SVC(probability=True)

modelBAG = BaggingClassifier(base_estimator=None, n_estimators=100,␣
↪max_samples=1.0, max_features=1.0,
                             bootstrap=True, bootstrap_features=False,␣
↪oob_score=False, warm_start=False,
                             n_jobs=None, random_state=None, verbose=0)

ModelGB = GradientBoostingClassifier(loss='deviance', learning_rate=0.1,␣
↪n_estimators=100, subsample=1.0,
                                     criterion='friedman_mse',␣
↪min_samples_split=2, min_samples_leaf=1,
                                     min_weight_fraction_leaf=0.0, max_depth=3,␣
↪min_impurity_decrease=0.0,
                                      init=None, random_state=None,
                                     max_features=None, verbose=0,␣
↪max_leaf_nodes=None, warm_start=False,
```

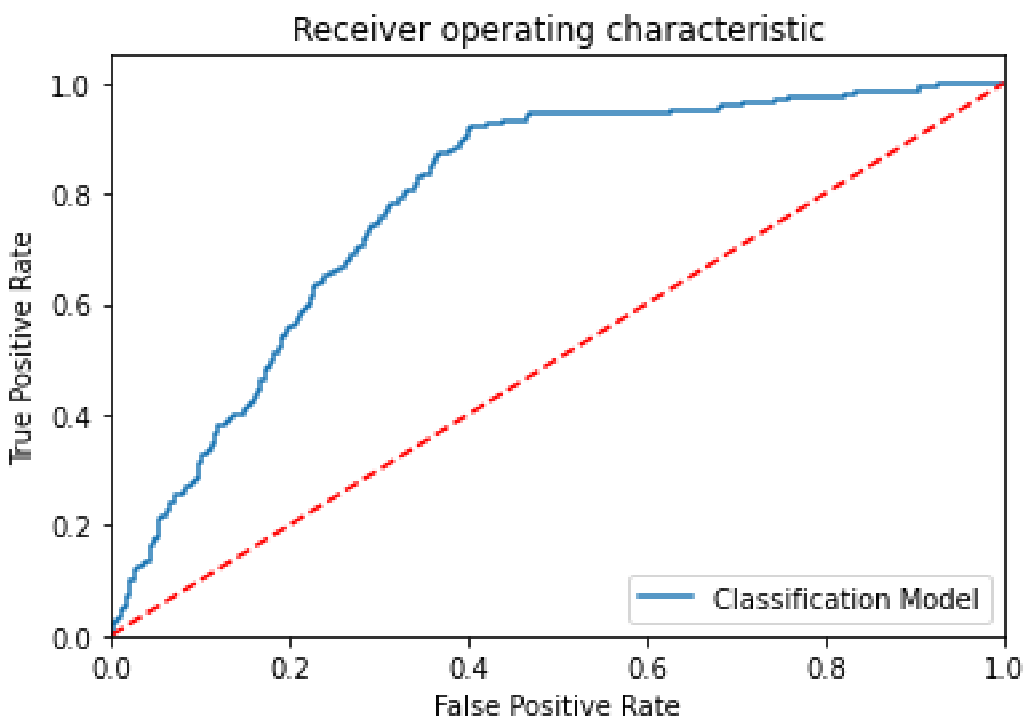
```
                                     validation_fraction=0.1,␣
↪n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
ModelLGB = lgb.LGBMClassifier()
ModelGNB = GaussianNB()

# Evalution matrix for all the algorithms

MM = [ModelLR, ModelDC, ModelRF, ModelET, ModelKNN, ModelSVM, modelBAG,␣
↪ModelGB, ModelLGB, ModelGNB]
for models in MM:

    # Fit the model

    models.fit(x_train, y_train)

    # Prediction

    y_pred = models.predict(x_test)
    y_pred_prob = models.predict_proba(x_test)

    # Print the model name

    print('Model Name: ', models)

    # confusion matrix in sklearn

    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import classification_report

    # actual values

    actual = y_test

    # predicted values

    predicted = y_pred

    # confusion matrix

    matrix = confusion_matrix(actual,predicted,␣
↪labels=[1,0],sample_weight=None, normalize=None)
    print('Confusion matrix : \n', matrix)

    # outcome values order in sklearn

    tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
    print('Outcome values : \n', tp, fn, fp, tn)
```

```python
# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie␣
↪between -1 to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%' ␣
↪)
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, models.predict_proba(x_test)[:,1])
```

```python
    plt.figure()
    # plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' %␣
↪logit_roc_auc)
    plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.savefig('Log_ROC')
    plt.show()
  ␣
↪print('-----------------------------------------------------------------------------------
  ␣
↪#-----------------------------------------------------------------------------------
    new_row = {'Model Name' : models,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
    Results = Results.append(new_row, ignore_index=True)
  ␣
↪#-----------------------------------------------------------------------------------
```

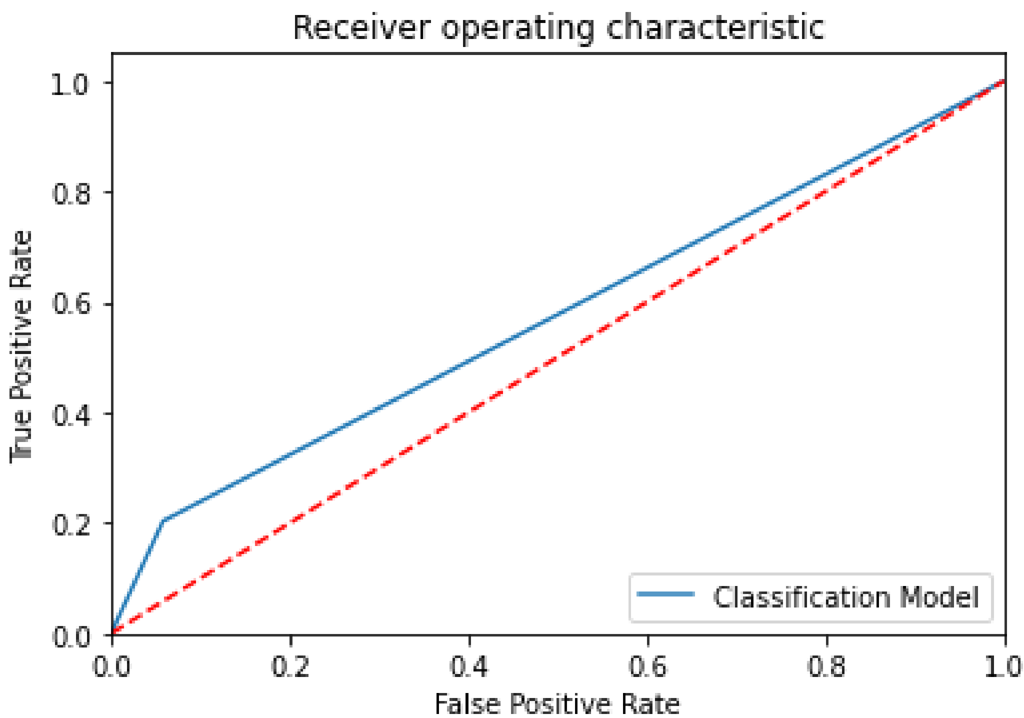
```
Model Name:  LogisticRegression()
Confusion matrix :
 [[   0  285]
 [   0 4341]]
Outcome values :
 0 285 0 4341
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
```

```
   macro avg       0.47      0.50      0.48      4626
weighted avg       0.88      0.94      0.91      4626

Accuracy : 93.8 %
Precision : nan %
Recall : 0.0 %
F1 Score : 0.0
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 50.0 %
MCC : nan
roc_auc_score: 0.5
```



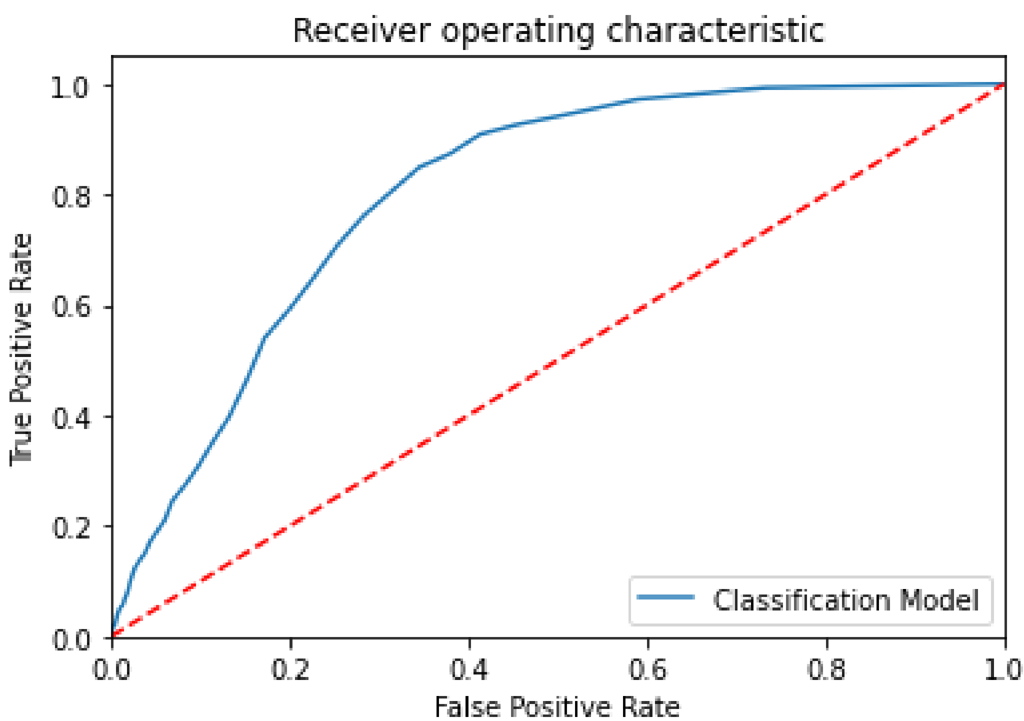

```
--------------------------------------------------------------------------------
---------------------
Model Name:  DecisionTreeClassifier()
Confusion matrix :
 [[  58  227]
 [ 253 4088]]
Outcome values :
 58 227 253 4088
Classification report :
               precision    recall  f1-score   support

           1       0.19      0.20      0.19       285
```

```
           0       0.95      0.94      0.94      4341

    accuracy                           0.90      4626
   macro avg       0.57      0.57      0.57      4626
weighted avg       0.90      0.90      0.90      4626

Accuracy : 89.6 %
Precision : 18.6 %
Recall : 20.4 %
F1 Score : 0.195
Specificity or True Negative Rate : 94.2 %
Balanced Accuracy : 57.3 %
MCC : 0.139
roc_auc_score: 0.573
```



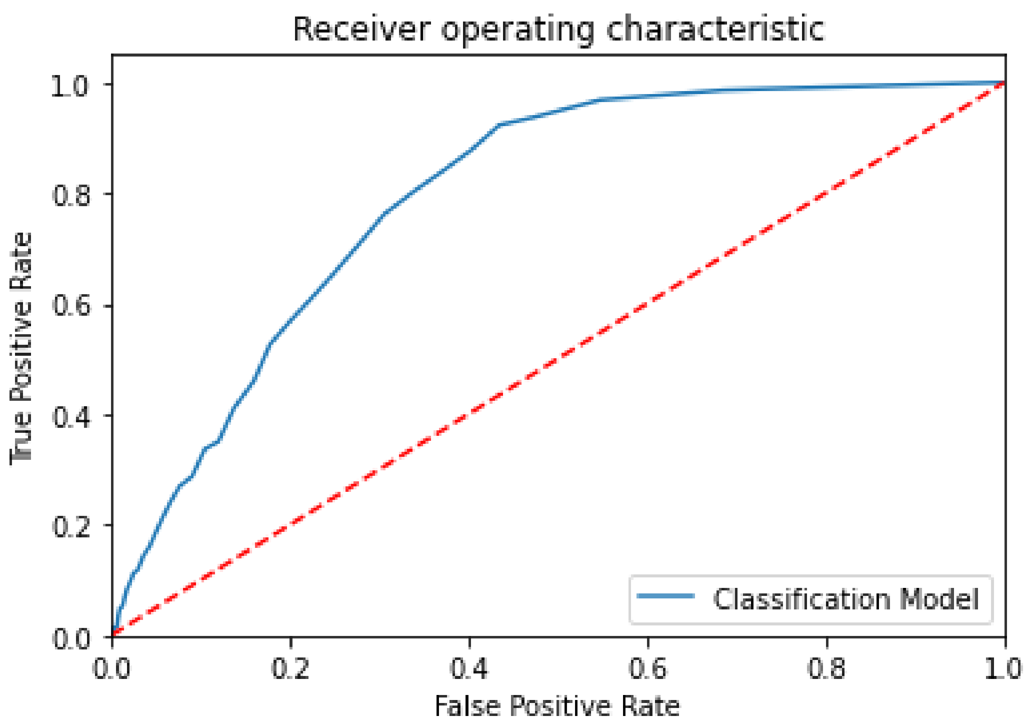

```
--------------------------------------------------------------------------------
---------------------
Model Name:  RandomForestClassifier()
Confusion matrix :
 [[   1  284]
 [   0 4341]]
Outcome values :
 1 284 0 4341
Classification report :
```

```
              precision    recall  f1-score   support

           1       1.00      0.00      0.01       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.97      0.50      0.49      4626
weighted avg       0.94      0.94      0.91      4626

Accuracy : 93.9 %
Precision : 100.0 %
Recall : 0.4 %
F1 Score : 0.007
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 50.2 %
MCC : 0.057
roc_auc_score: 0.502
```



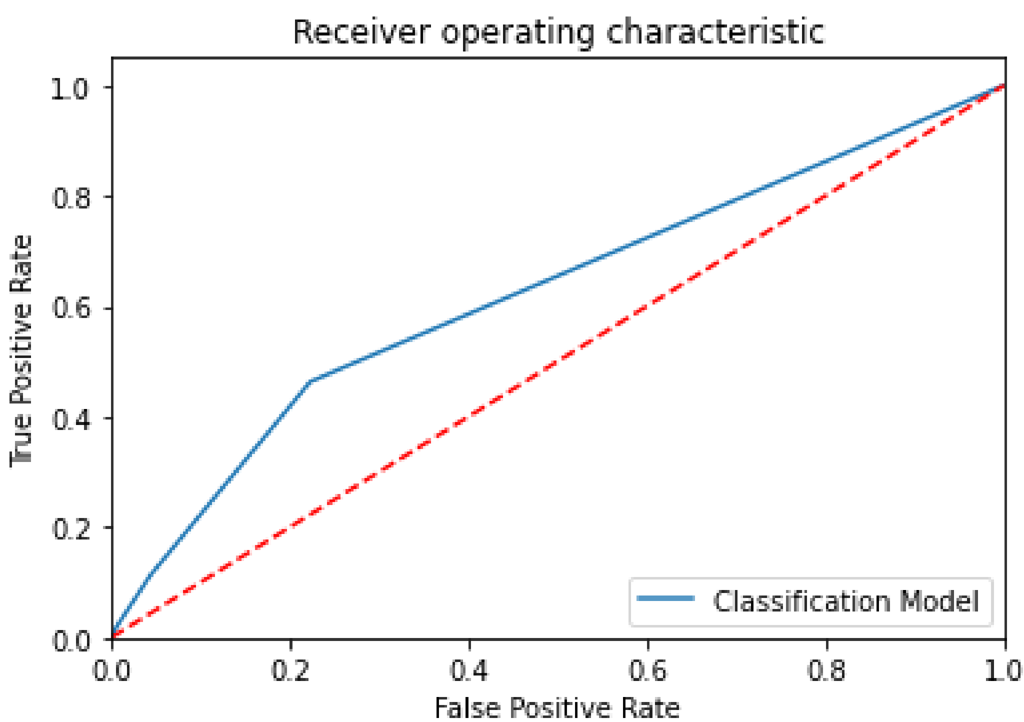

```
--------------------------------------------------------------------------------
---------------------
Model Name:  ExtraTreesClassifier()
Confusion matrix :
 [[   2  283]
 [   3 4338]]
```

```
Outcome values :
 2 283 3 4338
Classification report :
               precision    recall  f1-score   support

           1       0.40      0.01      0.01       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.67      0.50      0.49      4626
weighted avg       0.91      0.94      0.91      4626

Accuracy : 93.8 %
Precision : 40.0 %
Recall : 0.7 %
F1 Score : 0.014
Specificity or True Negative Rate : 99.9 %
Balanced Accuracy : 50.3 %
MCC : 0.046
roc_auc_score: 0.503
```

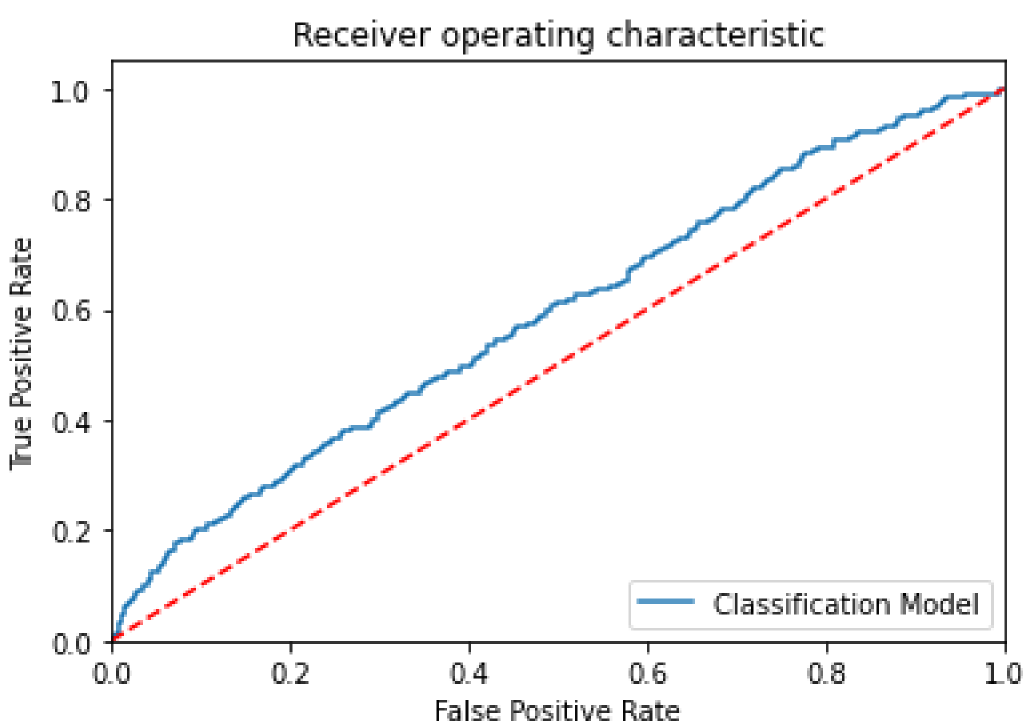



```
--------------------------------------------------------------------------------
---------------------
Model Name:  KNeighborsClassifier()
```

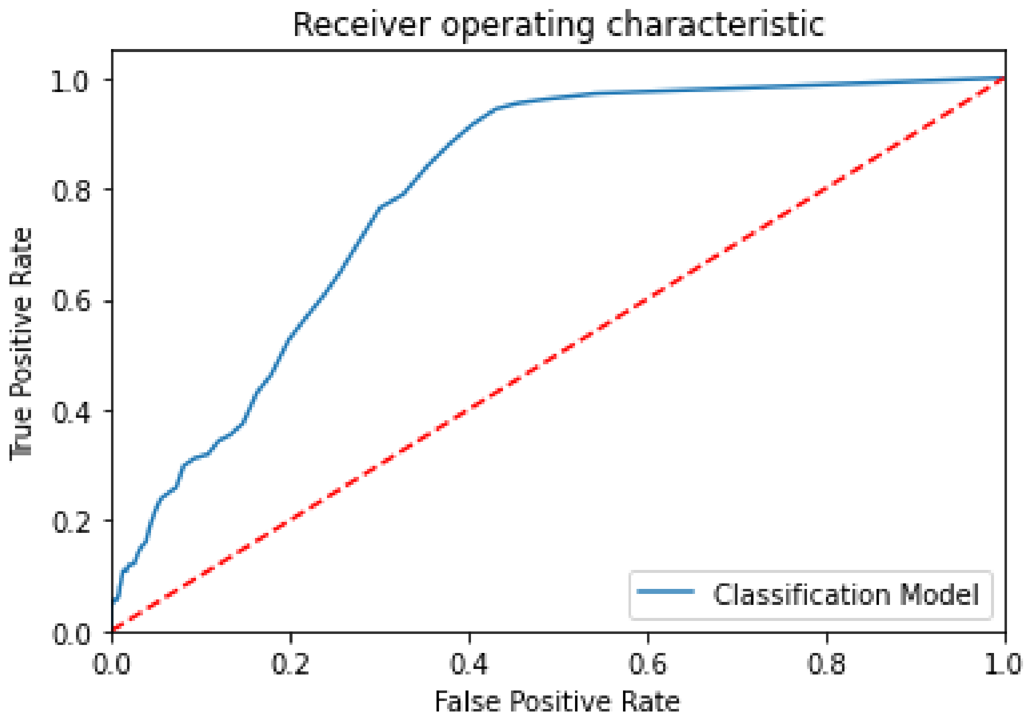
```
Confusion matrix :
 [[   5  280]
 [  23 4318]]
Outcome values :
 5 280 23 4318
Classification report :
               precision    recall  f1-score   support

           1       0.18      0.02      0.03       285
           0       0.94      0.99      0.97      4341

    accuracy                           0.93      4626
   macro avg       0.56      0.51      0.50      4626
weighted avg       0.89      0.93      0.91      4626

Accuracy : 93.5 %
Precision : 17.9 %
Recall : 1.8 %
F1 Score : 0.032
Specificity or True Negative Rate : 99.5 %
Balanced Accuracy : 50.6 %
MCC : 0.038
roc_auc_score: 0.506
```



```
--------------------------------------------------------------------------------
---------------------
Model Name:  SVC(probability=True)
Confusion matrix :
 [[   0  285]
 [   0 4341]]
Outcome values :
 0 285 0 4341
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.47      0.50      0.48      4626
weighted avg       0.88      0.94      0.91      4626

Accuracy : 93.8 %
Precision : nan %
Recall : 0.0 %
F1 Score : 0.0
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 50.0 %
MCC : nan
roc_auc_score: 0.5
```

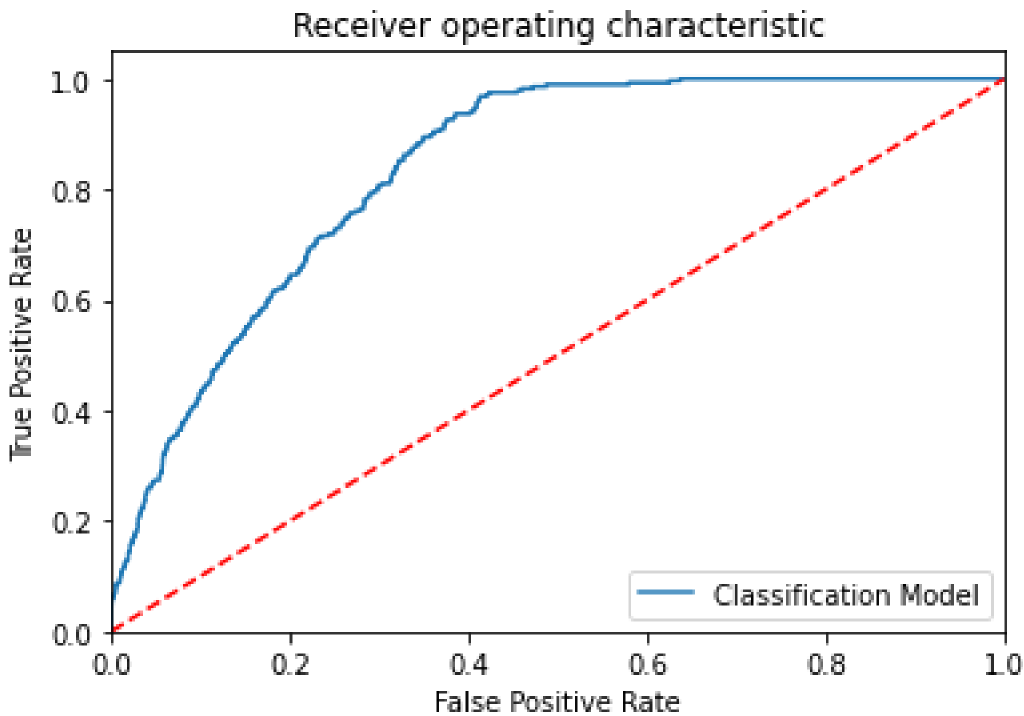


```
--------------------------------------------------------------------------------
---------------------
Model Name:  BaggingClassifier(n_estimators=100)
Confusion matrix :
 [[  14  271]
 [   7 4334]]
Outcome values :
 14 271 7 4334
Classification report :
              precision    recall  f1-score   support

           1       0.67      0.05      0.09       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.80      0.52      0.53      4626
weighted avg       0.92      0.94      0.91      4626

Accuracy : 94.0 %
Precision : 66.7 %
Recall : 4.9 %
F1 Score : 0.092
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 52.4 %
MCC : 0.17
roc_auc_score: 0.524
```



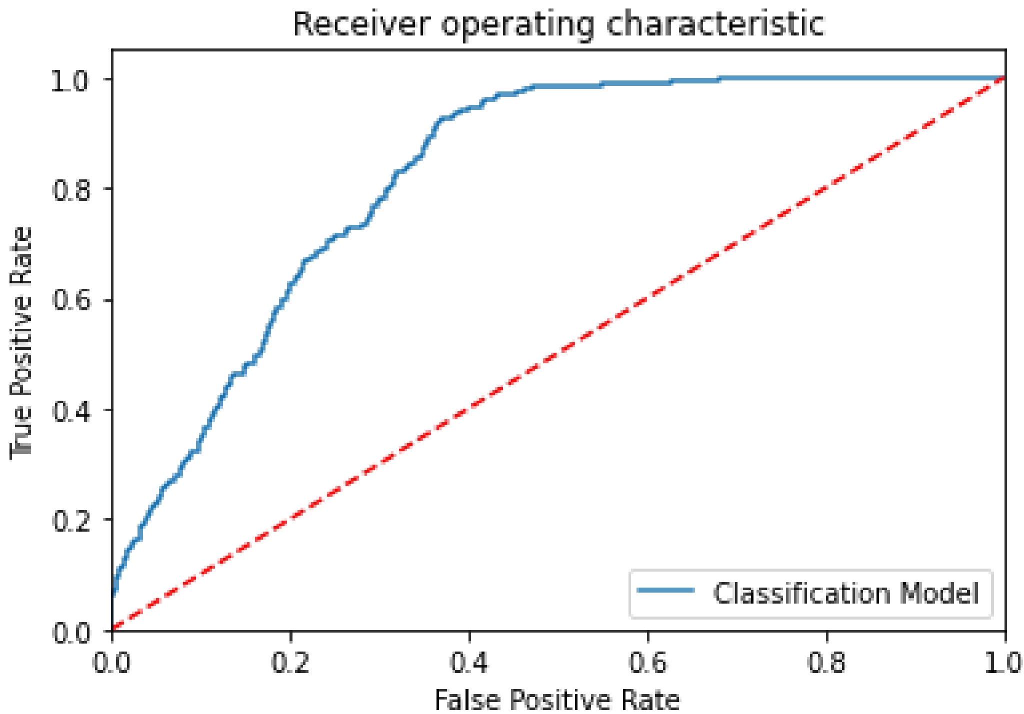

```
--------------------------------------------------------------------------------
---------------------
Model Name:  GradientBoostingClassifier(loss='deviance')
Confusion matrix :
 [[   1  284]
 [   2 4339]]
Outcome values :
 1 284 2 4339
Classification report :
               precision    recall  f1-score   support

           1       0.33      0.00      0.01       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.64      0.50      0.49      4626
weighted avg       0.90      0.94      0.91      4626

Accuracy : 93.8 %
Precision : 33.3 %
Recall : 0.4 %
F1 Score : 0.007
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 50.2 %
```

```
MCC : 0.029
roc_auc_score: 0.502
```



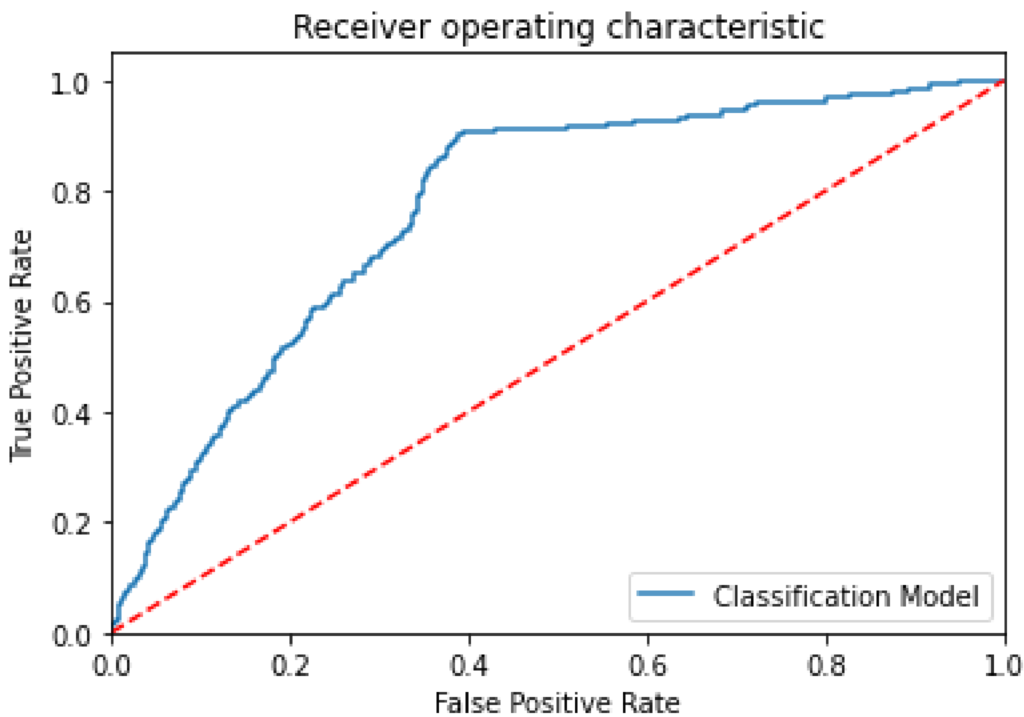

```
--------------------------------------------------------------------------------
---------------------
Model Name:  LGBMClassifier()
Confusion matrix :
 [[  16  269]
 [   3 4338]]
Outcome values :
 16 269 3 4338
Classification report :
               precision    recall  f1-score   support

           1       0.84      0.06      0.11       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.89      0.53      0.54      4626
weighted avg       0.94      0.94      0.92      4626

Accuracy : 94.1 %
Precision : 84.2 %
Recall : 5.6 %
```

```
F1 Score : 0.105
Specificity or True Negative Rate : 99.9 %
Balanced Accuracy : 52.8 %
MCC : 0.208
roc_auc_score: 0.528
```




```
--------------------------------------------------------------------------------
---------------------
Model Name:  GaussianNB()
Confusion matrix :
 [[ 117  168]
 [ 611 3730]]
Outcome values :
 117 168 611 3730
Classification report :
               precision    recall  f1-score   support

           1       0.16      0.41      0.23       285
           0       0.96      0.86      0.91      4341

    accuracy                           0.83      4626
   macro avg       0.56      0.63      0.57      4626
weighted avg       0.91      0.83      0.86      4626
```

```
Accuracy : 83.2 %
Precision : 16.1 %
Recall : 41.1 %
F1 Score : 0.231
Specificity or True Negative Rate : 85.9 %
Balanced Accuracy : 63.5 %
MCC : 0.178
roc_auc_score: 0.635
```




```
--------------------------------------------------------------------------------
--------------------
```

[18]:
```
# Results with comparing the all the algorithms

#Results.to_csv("/home/lab1/Downloads/Results.csv")

Results.head(20)
```

[18]:
```
                                           Model Name True_Positive  \
0                                  LogisticRegression()             0
1                              DecisionTreeClassifier()            58
2  (DecisionTreeClassifier(max_features='sqrt', r…            1
3  (ExtraTreeClassifier(random_state=596646805), …            2
4                                KNeighborsClassifier()             5
```

```
5                                  SVC(probability=True)           0
6  (DecisionTreeClassifier(random_state=274417190…           14
7  ([DecisionTreeRegressor(criterion='friedman_ms…            1
8                                       LGBMClassifier()          16
9                                           GaussianNB()         117

  False_Negative False_Positive True_Negative Accuracy Precision Recall  \
0            285              0          4341    0.938       NaN    0.0
1            227            253          4088    0.896     0.186  0.204
2            284              0          4341    0.939       1.0  0.004
3            283              3          4338    0.938       0.4  0.007
4            280             23          4318    0.935     0.179  0.018
5            285              0          4341    0.938       NaN    0.0
6            271              7          4334     0.94     0.667  0.049
7            284              2          4339    0.938     0.333  0.004
8            269              3          4338    0.941     0.842  0.056
9            168            611          3730    0.832     0.161  0.411

  F1 Score Specificity    MCC ROC_AUC_Score Balanced Accuracy
0      0.0         1.0    NaN           0.5               0.5
1    0.195       0.942  0.139      0.572614             0.573
2    0.007         1.0  0.057      0.501754             0.502
3    0.014       0.999  0.046      0.503163             0.503
4    0.032       0.995  0.038      0.506123             0.506
5      0.0         1.0    NaN           0.5               0.5
6    0.092       0.998   0.17      0.523755             0.524
7    0.007         1.0  0.029      0.501524             0.502
8    0.105       0.999  0.208      0.527725             0.528
9    0.231       0.859  0.178      0.634888             0.635
```

By observing these results we can say the top three best alogorithms are

1. LGBM Classification
2. Bagging Classification
3. Gradient Boosting Classification

1 HYPER PARAMETRIC TUNING

2 Hyper parameter tuning for LGBM classifier

2.1 case:01

GS_grid = { 'boosting_type':['gbdt','dart','goss','rf'] }

[19]:
```
# Hyper parameter tuning for lgbm classifier using grid search

from sklearn.model_selection import GridSearchCV
```

```
# Create the parameter grid based on the results of random search

GS_grid = {
           'boosting_type':['gbdt','dart','goss','rf']
          }

# Create object for model

ModelLGB = lgb.LGBMClassifier()

# Instantiate the grid search model

Grid_search = GridSearchCV(estimator =ModelLGB , param_grid = GS_grid, cv = 3,␣
 ↪n_jobs = -1, verbose = 2)

# Fit the grid search to the data

Grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
```

```
[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .
```

[19]:
```
GridSearchCV(cv=3, estimator=LGBMClassifier(), n_jobs=-1,
             param_grid={'boosting_type': ['gbdt', 'dart', 'goss', 'rf']},
             verbose=2)
```

[20]:
```
# Best parameter from gridseachCV

Grid_search.best_params_
```

[20]:
```
{'boosting_type': 'dart'}
```

[21]:
```
# Evalution matrix for the algorithms
ModelLGB=lgb.LGBMClassifier(boosting_type='dart', num_leaves=31, max_depth=-1,␣
 ↪learning_rate=0.1,
```

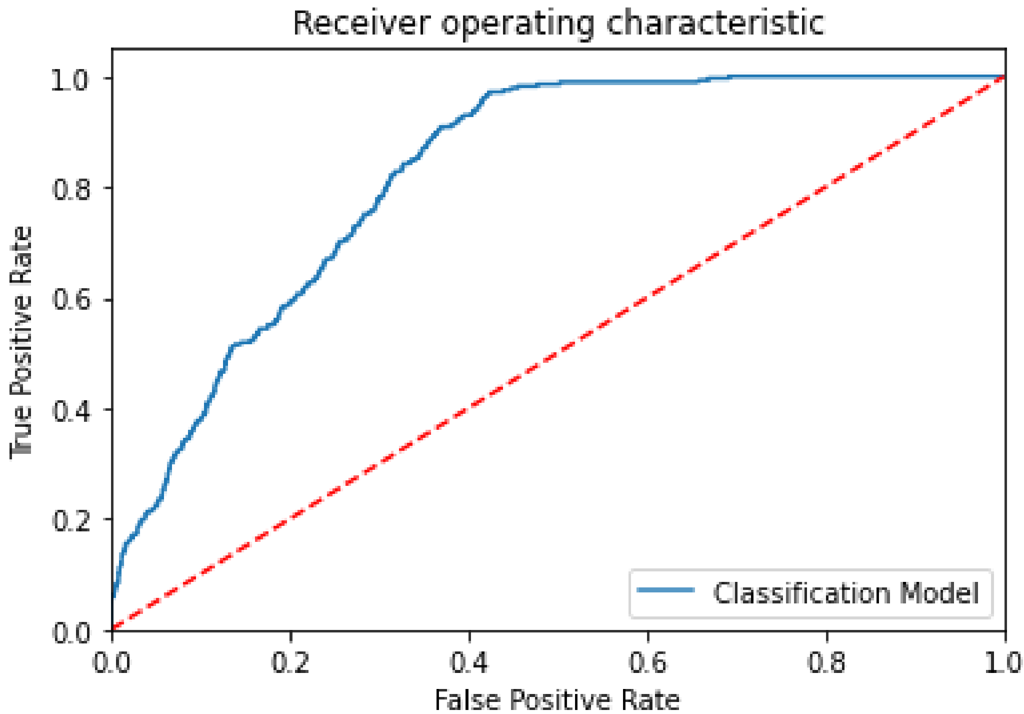
```
                n_estimators=100, subsample_for_bin=200000, objective=None,␣
 ↪class_weight=None,
                min_split_gain=0.0, min_child_weight=0.001,␣
 ↪min_child_samples=20, subsample=1.0,
                subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,␣
 ↪reg_lambda=0.0,
                random_state=None, n_jobs=None, importance_type='split')
# Fit the model

ModelLGB.fit(x_train, y_train)

# Prediction

y_pred = ModelLGB.predict(x_test)
y_pred_prob = ModelLGB.predict_proba(x_test)

# Print the model name

print('Model Name: ', models)

# confusion matrix in sklearn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,␣
 ↪normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])
```

```python
print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1␣
↪to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
```

```
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-----------------------------------------------------------------------------------------
#-----------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelLGB,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
Results = Results.append(new_row, ignore_index=True)
```

```
Model Name:  GaussianNB()
Confusion matrix :
 [[  14  271]
 [   0 4341]]
Outcome values :
 14 271 0 4341
Classification report :
               precision    recall  f1-score   support

           1       1.00      0.05      0.09       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.97      0.52      0.53      4626
weighted avg       0.94      0.94      0.92      4626

Accuracy : 94.1 %
Precision : 100.0 %
Recall : 4.9 %
F1 Score : 0.094
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 52.4 %
MCC : 0.215
```

```
roc_auc_score: 0.525
```




```
-------------------------------------------------------------------------------
---------------------
```

2.2 case:02

GS_grid = { 'min_child_samples' : [10,20,30,40,50] }

[22]:
```
# Hyper parameter tuning for lgbm classifier using grid search

from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search

GS_grid = {
            'min_child_samples' : [10,20,30,40]
          }

# Create object for model

ModelLGB = lgb.LGBMClassifier()

# Instantiate the grid search model
```

```
Grid_search = GridSearchCV(estimator =ModelLGB , param_grid = GS_grid, cv = 3,␣
 ↪n_jobs = -1, verbose = 2)

# Fit the grid search to the data

Grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
```

[22]:
```
GridSearchCV(cv=3, estimator=LGBMClassifier(), n_jobs=-1,
             param_grid={'min_child_samples': [10, 20, 30, 40]}, verbose=2)
```

[23]:
```
# Best parameter from gridseachCV

Grid_search.best_params_
```

[23]:
```
{'min_child_samples': 20}
```

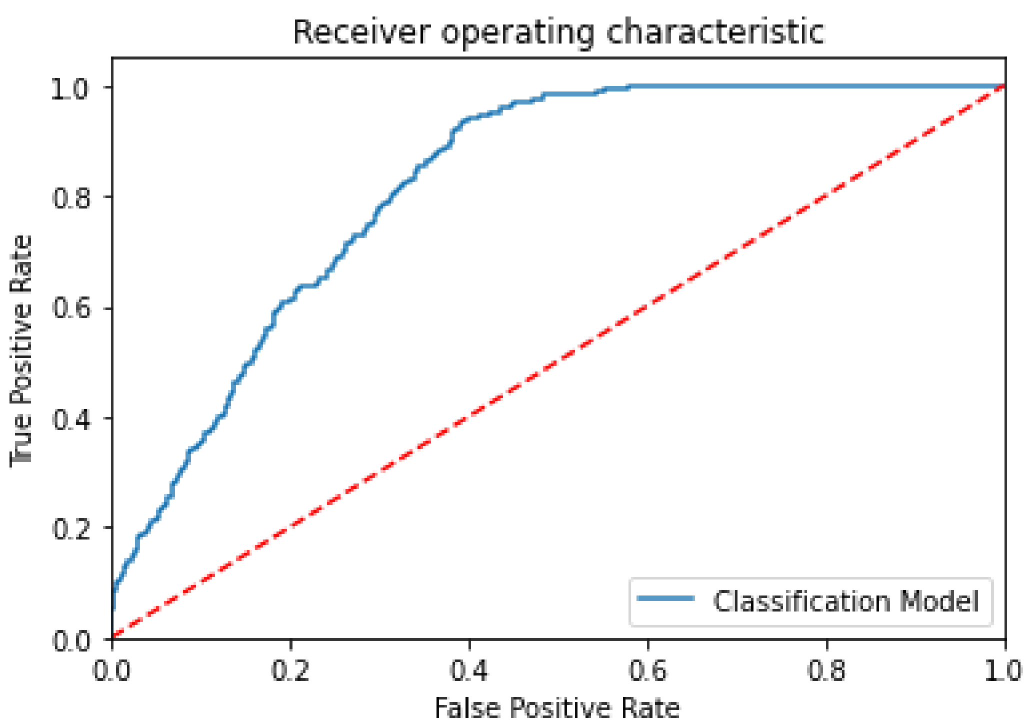
[24]:
```
# Evalution matrix for the algorithms
ModelLGB=lgb.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=-1,␣
 ↪learning_rate=0.1,
               n_estimators=200, subsample_for_bin=200000, objective=None,␣
 ↪class_weight=None,
               min_split_gain=0.0, min_child_weight=0.001,␣
 ↪min_child_samples=20, subsample=1.0,
               subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,␣
 ↪reg_lambda=0.0,
               random_state=None, n_jobs=None, importance_type='split')
# Fit the model

ModelLGB.fit(x_train, y_train)

# Prediction

y_pred = ModelLGB.predict(x_test)
y_pred_prob = ModelLGB.predict_proba(x_test)

# Print the model name

print('Model Name: ', models)

# confusion matrix in sklearn

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

```python
# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,␣
 ↪normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1␣
 ↪to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
```

```
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-----------------------------------------------------------------------------------
#----------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelLGB,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
Results = Results.append(new_row, ignore_index=True)
```

```
Model Name:  GaussianNB()
```

```
Confusion matrix :
 [[  15  270]
 [   6 4335]]
Outcome values :
 15 270 6 4335
Classification report :
               precision    recall  f1-score   support

           1       0.71      0.05      0.10       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.83      0.53      0.53      4626
weighted avg       0.93      0.94      0.92      4626

Accuracy : 94.0 %
Precision : 71.4 %
Recall : 5.3 %
F1 Score : 0.098
Specificity or True Negative Rate : 99.9 %
Balanced Accuracy : 52.6 %
MCC : 0.183
roc_auc_score: 0.526
```



2.3 case:03

GS_grid = { 'n_estimators': [100, 200, 300, 400, 500] }

[25]:
```
# Hyper parameter tuning for lgbm classifier using grid search

from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search

GS_grid = {
           'n_estimators': [100, 200, 300, 400, 500]
          }

# Create object for model

ModelLGB = lgb.LGBMClassifier()

# Instantiate the grid search model

Grid_search = GridSearchCV(estimator =ModelLGB , param_grid = GS_grid, cv = 3,␣
 ↪n_jobs = -1, verbose = 2)

# Fit the grid search to the data

Grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
```

[25]:
```
GridSearchCV(cv=3, estimator=LGBMClassifier(), n_jobs=-1,
             param_grid={'n_estimators': [100, 200, 300, 400, 500]}, verbose=2)
```

[26]:
```
# Best parameter from gridseachCV

Grid_search.best_params_
```

[26]:
```
{'n_estimators': 100}
```

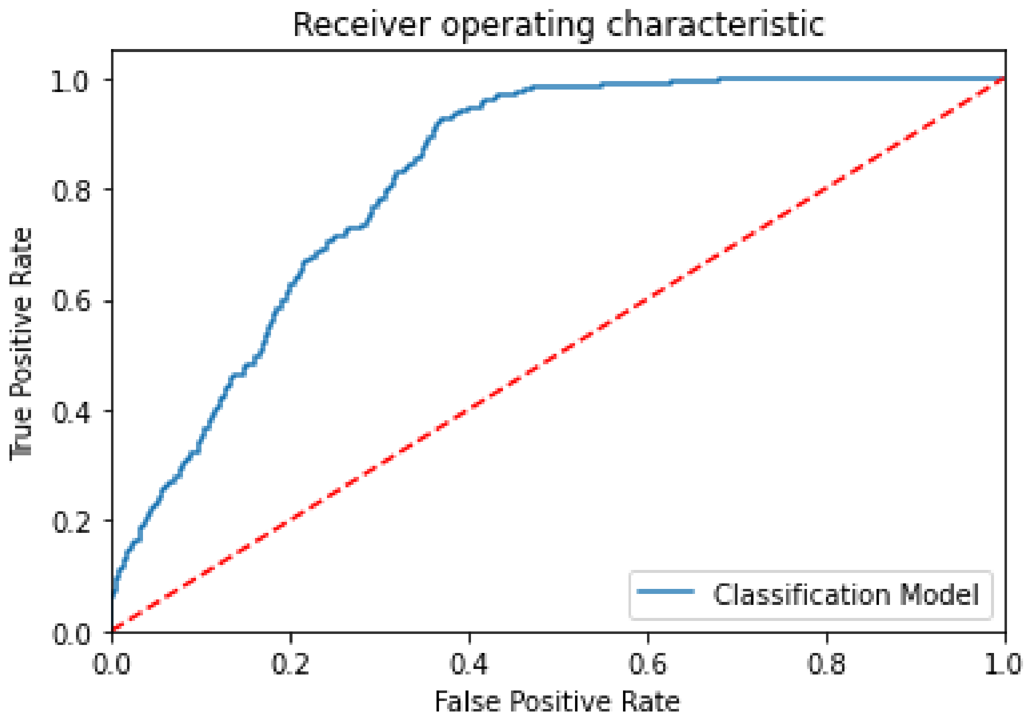
[27]:
```
# Evalution matrix for the algorithms
ModelLGB=lgb.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=-1,␣
 ↪learning_rate=0.1,
               n_estimators=100, subsample_for_bin=200000, objective=None,␣
 ↪class_weight=None,
               min_split_gain=0.0, min_child_weight=0.001,␣
 ↪min_child_samples=20, subsample=1.0,
```

```
                subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,␣
 ↪reg_lambda=0.0,
                random_state=None, n_jobs=None, importance_type='split')
# Fit the model

ModelLGB.fit(x_train, y_train)

# Prediction

y_pred = ModelLGB.predict(x_test)
y_pred_prob = ModelLGB.predict_proba(x_test)

# Print the model name

print('Model Name: ', models)

# confusion matrix in sklearn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,␣
 ↪normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics
```

```
sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1␣
 ↪to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
```

```python
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-----------------------------------------------------------------------------------------
#----------------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelLGB,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
Results = Results.append(new_row, ignore_index=True)
```

```
Model Name:  GaussianNB()
Confusion matrix :
 [[  16  269]
 [   3 4338]]
Outcome values :
 16 269 3 4338
Classification report :
               precision    recall  f1-score   support

           1       0.84      0.06      0.11       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.89      0.53      0.54      4626
weighted avg       0.94      0.94      0.92      4626

Accuracy : 94.1 %
Precision : 84.2 %
Recall : 5.6 %
F1 Score : 0.105
Specificity or True Negative Rate : 99.9 %
Balanced Accuracy : 52.8 %
MCC : 0.208
roc_auc_score: 0.528
```



2.4 case:04

GS_grid = { 'importance_type':['split','gain'] }

[28]:
```
# Hyper parameter tuning for lgbm classifier using grid search
GS_grid = {
            'importance_type':['split','gain']
          }
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search

GS_grid = {
            'importance_type':['split','gain']
          }

# Create object for model

ModelLGB = lgb.LGBMClassifier()

# Instantiate the grid search model
```

```
Grid_search = GridSearchCV(estimator =ModelLGB , param_grid = GS_grid, cv = 3,␣
↪n_jobs = -1, verbose = 2)

# Fit the grid search to the data

Grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
```

[28]:
```
GridSearchCV(cv=3, estimator=LGBMClassifier(), n_jobs=-1,
             param_grid={'importance_type': ['split', 'gain']}, verbose=2)
```

[29]:
```
# Best parameter from gridseachCV

Grid_search.best_params_
```

[29]:
```
{'importance_type': 'split'}
```

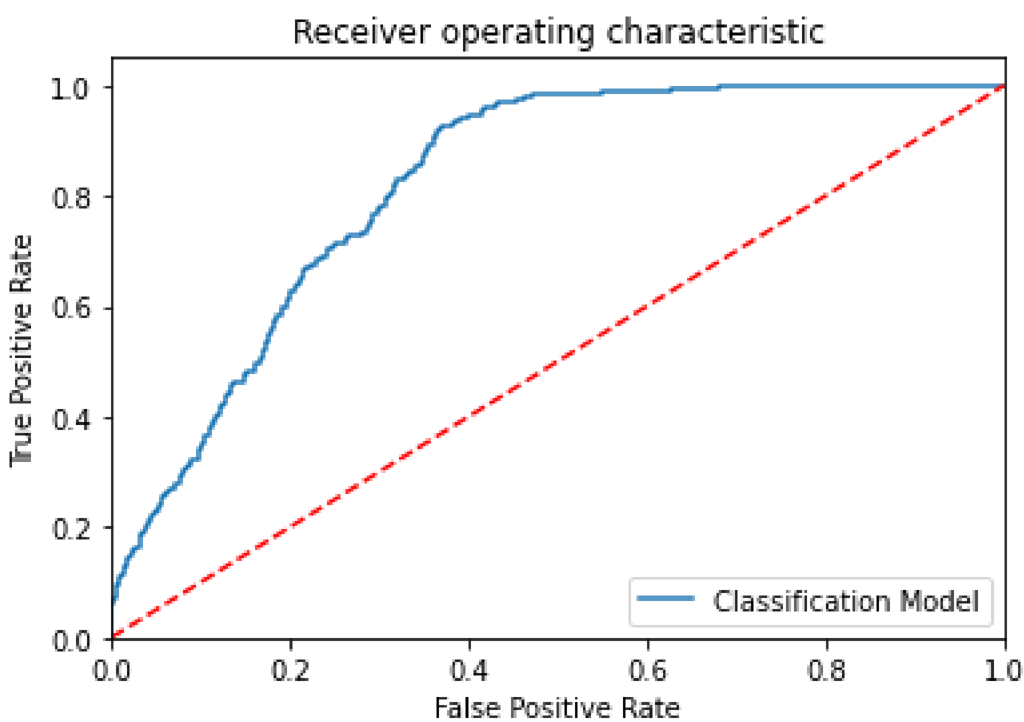
[30]:
```
# Evalution matrix for the algorithms
ModelLGB=lgb.LGBMClassifier(boosting_type='gbdt', num_leaves=31, max_depth=-1,␣
↪learning_rate=0.1,
               n_estimators=100, subsample_for_bin=200000, objective=None,␣
↪class_weight=None,
               min_split_gain=0.0, min_child_weight=0.001,␣
↪min_child_samples=20, subsample=1.0,
               subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,␣
↪reg_lambda=0.0,
               random_state=None, n_jobs=None, importance_type='split')
# Fit the model

ModelLGB.fit(x_train, y_train)

# Prediction

y_pred = ModelLGB.predict(x_test)
y_pred_prob = ModelLGB.predict_proba(x_test)

# Print the model name

print('Model Name: ', models)

# confusion matrix in sklearn

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

```python
# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix

matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,␣
↪normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1␣
↪to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
```

```
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('---------------------------------------------------------------------------------------
#-----------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelLGB,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
Results = Results.append(new_row, ignore_index=True)
```

```
Model Name:  GaussianNB()
```

```
Confusion matrix :
 [[  16  269]
 [   3 4338]]
Outcome values :
 16 269 3 4338
Classification report :
              precision    recall  f1-score   support

           1       0.84      0.06      0.11       285
           0       0.94      1.00      0.97      4341

    accuracy                           0.94      4626
   macro avg       0.89      0.53      0.54      4626
weighted avg       0.94      0.94      0.92      4626

Accuracy : 94.1 %
Precision : 84.2 %
Recall : 5.6 %
F1 Score : 0.105
Specificity or True Negative Rate : 99.9 %
Balanced Accuracy : 52.8 %
MCC : 0.208
roc_auc_score: 0.528
```



2.5 case:05

GS_grid = { 'boosting_type':['gbdt','dart','goss','rf'], 'min_child_samples' : [10,20,30,40,50], 'n_estimators': [100, 200, 300, 400, 500], 'importance_type':['split','gain'] }

[31]:
```python
# Hyper parameter tuning for lgbm classifier using grid search

from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search

GS_grid = {
           'boosting_type':['gbdt','dart','goss','rf'],
           'min_child_samples' : [10,20,30,40],
           'n_estimators': [100, 200, 300, 400, 500],
           'importance_type':['split','gain']
          }

# Create object for model

ModelLGB = lgb.LGBMClassifier()

# Instantiate the grid search model

Grid_search = GridSearchCV(estimator =ModelLGB , param_grid = GS_grid, cv = 3,␣
 ↪n_jobs = -1, verbose = 2)

# Fit the grid search to the data

Grid_search.fit(x_train,y_train)
```

```
Fitting 3 folds for each of 160 candidates, totalling 480 fits

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[CV] END …boosting_type=dart; total time=   0.3s
[CV] END …boosting_type=rf; total time=   0.0s
[CV] END …min_child_samples=20; total time=   0.1s
[CV] END …min_child_samples=40; total time=   0.1s
```

```
[CV] END …n_estimators=200; total time=   0.2s
[CV] END …n_estimators=400; total time=   0.4s
[CV] END …importance_type=split; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=300; total time=   0.4s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.7s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
```

```
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   1.6s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   1.6s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   1.7s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=500; total time=   3.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=400; total time=   2.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=300; total time=   1.4s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=500; total time=   3.4s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   2.5s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=300; total time=   1.7s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=500; total time=   3.9s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=400; total time=   2.8s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
```

```
n_estimators=100; total time=   0.1s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   0.4s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.7s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.7s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=500; total time=   0.8s

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .
```

```
[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .
```

```
[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .
```

```
[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .


[CV] END …boosting_type=gbdt; total time=   0.1s
[CV] END …boosting_type=goss; total time=   0.2s
[CV] END …min_child_samples=20; total time=   0.1s
```

```
[CV] END …min_child_samples=40; total time=   0.1s
[CV] END …n_estimators=200; total time=   0.2s
[CV] END …n_estimators=400; total time=   0.4s
[CV] END …importance_type=split; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=20,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   0.6s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=20,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=20,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   1.5s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=400; total time=   2.4s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=400; total time=   2.6s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=20,
n_estimators=400; total time=   2.5s
```

```
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   2.7s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   1.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   4.1s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   2.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=300; total time=   1.5s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=500; total time=   3.5s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=400; total time=   2.7s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=300; total time=   1.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=500; total time=   4.1s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   0.5s
```

```
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=40,
n_estimators=400; total time=   0.7s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=400; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=400; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.0s
[CV] END …boosting_type=dart; total time=   0.3s
[CV] END …boosting_type=rf; total time=   0.0s
```

```
[CV] END …min_child_samples=10; total time=   0.1s
[CV] END …min_child_samples=30; total time=   0.1s
[CV] END …n_estimators=100; total time=   0.1s
[CV] END …n_estimators=300; total time=   0.3s
[CV] END …n_estimators=500; total time=   0.5s
[CV] END …importance_type=gain; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=10,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   0.3s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=100; total time=   0.1s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.2s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=split, min_child_samples=40,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=30,
n_estimators=400; total time=   0.4s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=gbdt, importance_type=gain, min_child_samples=40,
n_estimators=500; total time=   0.5s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   3.4s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   3.2s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   1.7s
```

```
[CV] END boosting_type=dart, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   2.7s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   1.8s
[CV] END boosting_type=dart, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   4.0s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   2.4s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=200; total time=   0.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=300; total time=   1.6s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=20,
n_estimators=500; total time=   3.6s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=30,
n_estimators=400; total time=   2.6s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=100; total time=   0.3s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=200; total time=   0.9s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=300; total time=   1.8s
[CV] END boosting_type=dart, importance_type=gain, min_child_samples=40,
n_estimators=500; total time=   4.0s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   0.4s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=30,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=split, min_child_samples=40,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=10,
n_estimators=400; total time=   0.6s
```

```
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=20,
n_estimators=400; total time=   0.6s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=100; total time=   0.2s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=200; total time=   0.3s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=300; total time=   0.5s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=30,
n_estimators=500; total time=   0.8s
[CV] END boosting_type=goss, importance_type=gain, min_child_samples=40,
n_estimators=400; total time=   0.7s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=300; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=400; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=10,
n_estimators=500; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=300; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=20,
n_estimators=500; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=100; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=200; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=300; total time=   0.0s
[CV] END boosting_type=rf, importance_type=split, min_child_samples=30,
n_estimators=400; total time=   0.0s

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
```

```
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .

[LightGBM] [Fatal] Check failed: config->bagging_freq > 0 &&
config->bagging_fraction < 1.0f && config->bagging_fraction > 0.0f at
/__w/1/s/python-package/compile/src/boosting/rf.hpp, line 35 .
```

```
[31]: GridSearchCV(cv=3, estimator=LGBMClassifier(), n_jobs=-1,
                   param_grid={'boosting_type': ['gbdt', 'dart', 'goss', 'rf'],
                               'importance_type': ['split', 'gain'],
                               'min_child_samples': [10, 20, 30, 40],
                               'n_estimators': [100, 200, 300, 400, 500]},
                   verbose=2)
```

[32]:
```
# Best parameter from gridseachCV

Grid_search.best_params_
```

[32]:
```
{'boosting_type': 'dart',
 'importance_type': 'split',
 'min_child_samples': 10,
 'n_estimators': 200}
```

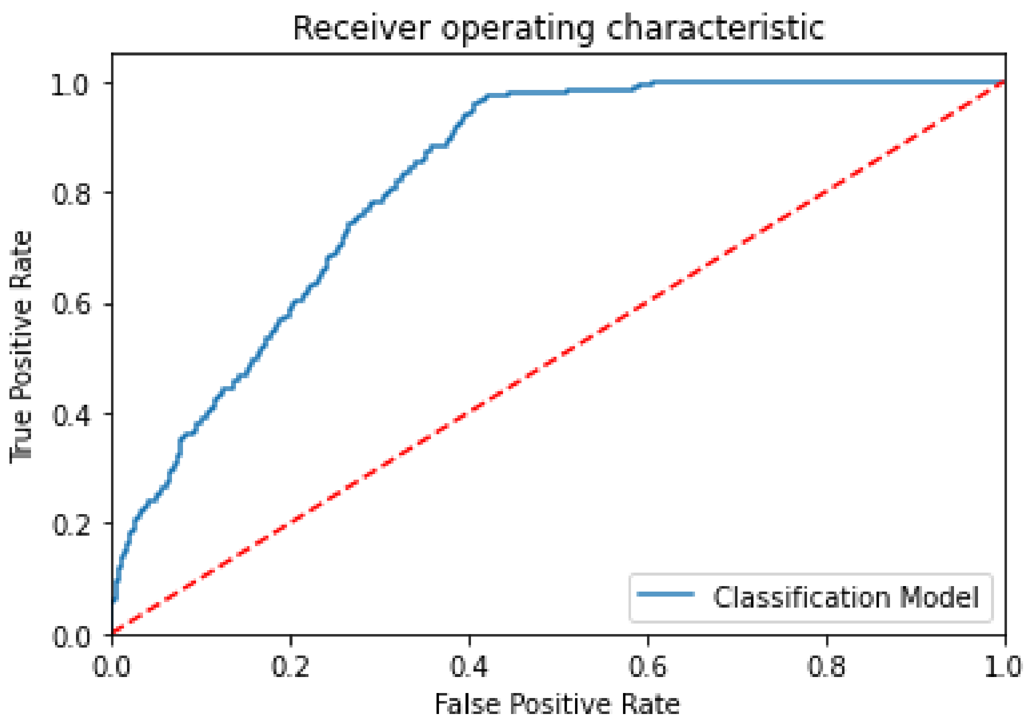
[33]:
```
# Evalution matrix for the algorithms
ModelLGB=lgb.LGBMClassifier(boosting_type='dart', num_leaves=31, max_depth=-1,␣
 ↪learning_rate=0.1,
               n_estimators=200, subsample_for_bin=200000, objective=None,␣
 ↪class_weight=None,
               min_split_gain=0.0, min_child_weight=0.001,␣
 ↪min_child_samples=10, subsample=1.0,
               subsample_freq=0, colsample_bytree=1.0, reg_alpha=0.0,␣
 ↪reg_lambda=0.0,
               random_state=None, n_jobs=None, importance_type='split')
# Fit the model

ModelLGB.fit(x_train, y_train)

# Prediction

y_pred = ModelLGB.predict(x_test)
y_pred_prob = ModelLGB.predict_proba(x_test)

# Print the model name

print('Model Name: ', models)

# confusion matrix in sklearn

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

# actual values

actual = y_test

# predicted values

predicted = y_pred

# confusion matrix
```

```
matrix = confusion_matrix(actual,predicted, labels=[1,0],sample_weight=None,␣
↪normalize=None)
print('Confusion matrix : \n', matrix)

# outcome values order in sklearn

tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# classification report for precision, recall f1-score and accuracy

C_Report = classification_report(actual,predicted,labels=[1,0])

print('Classification report : \n', C_Report)

# calculating the metrics

sensitivity = round(tp/(tp+fn), 3);
specificity = round(tn/(tn+fp), 3);
accuracy = round((tp+tn)/(tp+fp+tn+fn), 3);
balanced_accuracy = round((sensitivity+specificity)/2, 3);

precision = round(tp/(tp+fp), 3);
f1Score = round((2*tp/(2*tp + fp + fn)), 3);

# Matthews Correlation Coefficient (MCC). Range of values of MCC lie between -1␣
↪to +1.
# A model with a score of +1 is a perfect model and -1 is a poor model

from math import sqrt

mx = (tp+fp) * (tp+fn) * (tn+fp) * (tn+fn)
MCC = round(((tp * tn) - (fp * fn)) / sqrt(mx), 3)

print('Accuracy :', round(accuracy*100, 2),'%')
print('Precision :', round(precision*100, 2),'%')
print('Recall :', round(sensitivity*100,2), '%')
print('F1 Score :', f1Score)
print('Specificity or True Negative Rate :', round(specificity*100,2), '%'  )
print('Balanced Accuracy :', round(balanced_accuracy*100, 2),'%')
print('MCC :', MCC)

# Area under ROC curve

from sklearn.metrics import roc_curve, roc_auc_score

print('roc_auc_score:', round(roc_auc_score(actual, predicted), 3))
```

```python
# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLGB.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot(fpr, tpr, label= 'Classification Model' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
print('-----------------------------------------------------------------------------------------
#-----------------------------------------------------------------------------------------
new_row = {'Model Name' : ModelLGB,
               'True_Positive' : tp,
               'False_Negative' : fn,
               'False_Positive' : fp,
               'True_Negative' : tn,
               'Accuracy' : accuracy,
               'Precision' : precision,
               'Recall' : sensitivity,
               'F1 Score' : f1Score,
               'Specificity' : specificity,
               'MCC':MCC,
               'ROC_AUC_Score':roc_auc_score(actual, predicted),
               'Balanced Accuracy':balanced_accuracy}
Results = Results.append(new_row, ignore_index=True)
```

```
Model Name:  GaussianNB()
Confusion matrix :
 [[  14  271]
 [   0 4341]]
Outcome values :
 14 271 0 4341
Classification report :
               precision    recall  f1-score   support

           1       1.00      0.05      0.09       285
           0       0.94      1.00      0.97      4341
```

```
    accuracy                           0.94      4626
   macro avg       0.97      0.52      0.53      4626
weighted avg       0.94      0.94      0.92      4626

Accuracy : 94.1 %
Precision : 100.0 %
Recall : 4.9 %
F1 Score : 0.094
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 52.4 %
MCC : 0.215
roc_auc_score: 0.525
```




```
--------------------------------------------------------------------------------
---------------------
```

[34]:
```
#Final Results

Results
```

[34]:
```
                                         Model Name True_Positive  \
0                              LogisticRegression()             0
1                          DecisionTreeClassifier()            58
```

2	(DecisionTreeClassifier(max_features='sqrt', r…	1
3	(ExtraTreeClassifier(random_state=596646805), …	2
4	KNeighborsClassifier()	5
5	SVC(probability=True)	0
6	(DecisionTreeClassifier(random_state=274417190…	14
7	([DecisionTreeRegressor(criterion='friedman_ms…	1
8	LGBMClassifier()	16
9	GaussianNB()	117
10	LGBMClassifier(boosting_type='dart', n_jobs=None)	14
11	LGBMClassifier(n_estimators=200, n_jobs=None)	15
12	LGBMClassifier(n_jobs=None)	16
13	LGBMClassifier(n_jobs=None)	16
14	LGBMClassifier(boosting_type='dart', min_child…	14

	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall \
0	285	0	4341	0.938	NaN	0.0
1	227	253	4088	0.896	0.186	0.204
2	284	0	4341	0.939	1.0	0.004
3	283	3	4338	0.938	0.4	0.007
4	280	23	4318	0.935	0.179	0.018
5	285	0	4341	0.938	NaN	0.0
6	271	7	4334	0.94	0.667	0.049
7	284	2	4339	0.938	0.333	0.004
8	269	3	4338	0.941	0.842	0.056
9	168	611	3730	0.832	0.161	0.411
10	271	0	4341	0.941	1.0	0.049
11	270	6	4335	0.94	0.714	0.053
12	269	3	4338	0.941	0.842	0.056
13	269	3	4338	0.941	0.842	0.056
14	271	0	4341	0.941	1.0	0.049

	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	0.0	1.0	NaN	0.5	0.5
1	0.195	0.942	0.139	0.572614	0.573
2	0.007	1.0	0.057	0.501754	0.502
3	0.014	0.999	0.046	0.503163	0.503
4	0.032	0.995	0.038	0.506123	0.506
5	0.0	1.0	NaN	0.5	0.5
6	0.092	0.998	0.17	0.523755	0.524
7	0.007	1.0	0.029	0.501524	0.502
8	0.105	0.999	0.208	0.527725	0.528
9	0.231	0.859	0.178	0.634888	0.635
10	0.094	1.0	0.215	0.524561	0.524
11	0.098	0.999	0.183	0.525625	0.526
12	0.105	0.999	0.208	0.527725	0.528
13	0.105	0.999	0.208	0.527725	0.528
14	0.094	1.0	0.215	0.524561	0.524

By concluding with Hyper-parametric tuning, There is no increment in the performance of model compared to actual model without hyper-parametric tuning