Table of Contents

Abstract

Today organizations, which hire data scientists are especially interested in job candidate’s portfolio. Analysis of organization’s marketing data is one of the most typical applications of data science and machine learning. Such Analysis will definitely be a nice contribution to this portfolio.

Data Collection: AI algorithms can be used to gather admission data from multiple IITs and NITs over the years. This data includes information about cutoff ranks, courses, and other relevant details.

Data Preprocessing: The collected data needs to be cleaned and structured for analysis. This includes handling missing values and removing any inconsistencies.

Feature Engineering: Relevant features such as the year of admission, course, category, and specific institute need to be identified and extracted.

This dataset containing IIT-NIT cut-off campaign data and we can use it to optimize marketing campaigns to attract more customers to a term deposit Subscription.

In order to optimize marketing campaigns with the help of a dataset, we will have to take following steps:

1. Import data from datasets and perform initial high level analysis
2. Clean the Data
3. Use Machine Learning Techniques

IIT-NIT cut-off process steps:

- Understand the student cut-offs
- Develop a basic strategy
- Make a Decision
- Execute a plan
- Deliver the results

CHAPTER 1
INTRODUCTION

With the increasing power of computer technology, companies and institutions can nowadays store large amounts of data at reduced cost. The amount of available data is increasing exponentially and cheap disk storage makes it easy to store data that previously was thrown away. There is a huge amount of information locked up in databases that is potentially important but has not yet been explored. The growing size and complexity of the databases makes it hard to analyse the data manually, so it is important to have automated systems to support the process. Hence there is the need of computational tools able to treat these large amounts of data and extract valuable information.

In this context, Data Mining provides automated systems capable of processing large amounts of data that are already present in databases. Data Mining is used to automatically extract important patterns and trends from databases seeking regularities or patterns that can reveal the structure of the data and answer business problems. Data mining includes learning techniques that fall into the field of Machine learning. Thegrowth of databases in recent years brings data mining at the forefront of new business technologies.

A key challenge for the insurance industry is to charge each customer an appropriate price for the risk they represent. Risk varies widely from customer to customer and a deep understanding of different risk factors helps predict the likelihood and cost of insurance claims. The goal of this program is to see how well various statistical methods perform in predicting auto Insurance claims based on the characteristics of the driver, vehicle and driver / vehicle coverage details.

A number of factors will determine BI claims prediction among them a driver's age, past accident history, and domicile, etc. However, this contest focused on the relationship between claims and vehicle characteristics well as other characteristics associated with the auto insurance policies.

1.1. What are the different types of Machine Learning?

How Machine Learning Works?

Machine learning uses two types of techniques:

Supervised learning, which trains a model on known input and output data so that it can predict future outputs, and Unsupervised learning, which finds hidden patterns or intrinsic structures in input data.

Supervised Learning:

Supervised machine learning builds a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data. Use supervised learning if you have known data for the output you are trying to predict.

Supervised learning uses Regression and Classification techniques to develop predictive models.

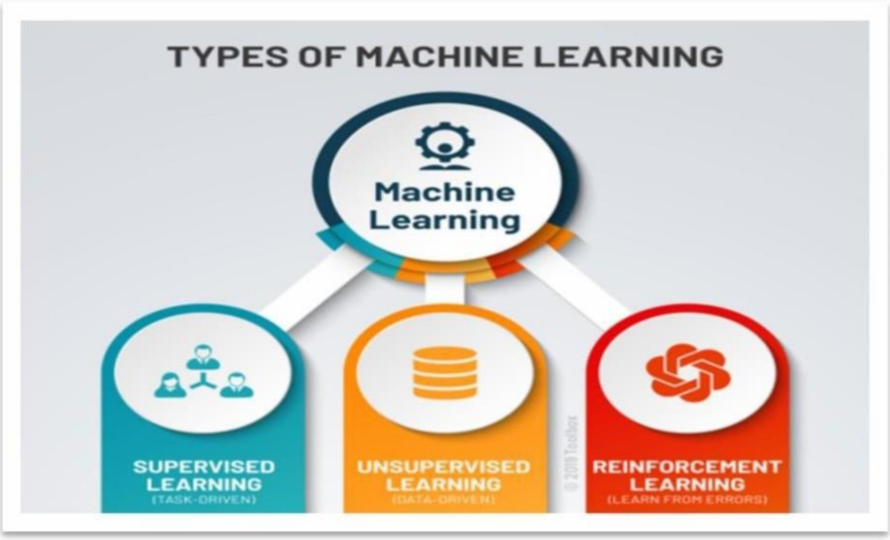


Fig 1.1 Types of machine learning

Regression techniques predict continuous responses - for example, changes in temperature or fluctuations in power demand. Typical applications include electricity load forecasting and algorithmic trading.

Use regression techniques if you are working with a data range or if the nature of your response is a real number, such as temperature or the time until failure for a piece of equipment.

Common regression algorithms include linear model, nonlinear model, stepwise regression, Gradient Descent Regression, Support Vector Regression, Ridge and Lasso Regressions.

Classification techniques predict discrete responses - for example, whether an email is genuine or spam, or whether a tumour is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring.

Use classification if your data can be tagged, categorized, or separated into specific groups or classes. For example, applications for hand-writing recognition use classification to recognize letters and numbers. In image processing and computer vision, unsupervised pattern recognition techniques are used for object detection and image segmentation.

Common algorithms for performing classification include support vector machine (SVM), boosted and bagged decision trees, k-nearest neighbour, Naïve Bayes, discriminant analysis, logistic regression, and neural networks.

Using Supervised Learning to Predict Heart Attacks: Suppose clinicians want to predict whether someone will have a heart attack within a year. They have data on previous patients, including age, weight, height, and blood pressure. They know whether the previous patients had heart attacks within a year. So, the problem is combining the existing data into a model that can predict whether a new person will have a heartattack within a year.

Unsupervised Learning:

Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data without labelled responses.

Clustering is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data. Applications for cluster analysis include gene sequence analysis and market research.

For example, if a cell phone company wants optimize the locations where they build cell phone towers, they can use machine learning to estimate the number of clusters of people relying on their towers. A phone can only talk to one tower at a time, so the team uses clustering algorithms to design the best placement of cell towers to optimize signal reception for groups, or clusters, of their customers.

Common algorithms for performing clustering include k-means and k-medoids, Apriori algorithms, hierarchical clustering, Gaussian mixture models and hidden Markov models.

1.2. Benefits of Using Machine Learning.

Exploring the Advantages and Disadvantages of Machine Learning:

- When it comes to learning technology, we should be aware of the pros and cons of that technology. The reason is so that we can understand the capabilities of that subject.

- That is exactly what we are doing here. Understanding the advantages and disadvantages of Machine Learning will help us to unlock many doors.

- The advantages of Machine Learning are vast. It helps us to create ways of modernizing technology. The disadvantages of Machine Learning tell us its limits and side effects. This helps us to find different innovative ways to reduce these problems.

Advantages of Machine Learning:

- **Automation of Everything:** Machine Learning is responsible for cutting the workload and time. By automating things, we let the algorithm do the hard work for us. Automation is now being done almost everywhere. The reason is that it is very reliable. Also, it helps us to think more creatively. Due to ML, we are now designing more advanced computers. These computers can handle various Machine Learning models and algorithms efficiently. Even though automation is spreading fast, we still don't completely rely on it. ML is slowly transforming the industry with its automation.
- **Wide Range of Applications:** ML has a wide variety of applications. This means that we can apply ML on any of the major fields. ML has its role everywhere from medical, business, banking to science and tech. This helps to create more opportunities. It plays a major role in customer interactions. Machine Learning can help in the detection of diseases more quickly. It is helping to lift up businesses. That is why investing in ML technology is worth it.
- **Scope of Improvement:** Machine Learning is the type of technology that keeps on evolving. There is a lot of scope in ML to become the top technology in the future. The reason is it has a lot of research areas in it. This helps us to improve both hardware and software. In hardware, we have various laptops and GPU. These have various ML and Deep

  Learning networks in them. These help in the faster processing power of the system. When it comes to software, we have various UI and libraries in use. These help in designing more efficient algorithms.
- **Efficient Handling of Data:** Machine Learning has many factors that make it reliable. One of them is data handling. ML plays the biggest role when it comes to data currently. It can handle any type of data. Machine Learning can be multidimensional or different types of data. It canprocess and analyse these data those normal systems can't. Data is the most important part of any Machine Learning model. Also, studying and handling of data is a field.
- **Best for Education and Online Shopping:**

ML would be the best tool for education in the future. It provides very creative techniques to help students study.

Recently in China, a school has started to use ML to improve student focus. In online shopping, the ML model studies your searches. Based on your search history, it would provide advertisements. These will be about your search preferences in previous searches. In this, the search history is the data for the model. This is a great way to improve e-commerce with ML.

AIML role in IIT -NIT cut-off ranks

Artificial Intelligence and Machine Learning (AIML) can play several significant roles in analyzing and predicting the cutoff ranks for admissions to IITs and NITs:

Data Analysis: AIML algorithms can analyze historical admission data, including cutoff ranks, student profiles, and institute-specific data. This analysis can reveal trends and patterns in the admission process.

Predictive Modeling: AIML models, such as regression and classification, can be applied to the data to predict future cutoff ranks. By considering factors like the number of applicants, available seats, and historical trends, these models can provide estimates of the expected cutoff ranks for upcoming admissions.

Rank Prediction: Students can use AI-powered tools to input their own academic and test score data to receive personalized predictions of the cutoff ranks required for their preferred courses and institutes. This helps students set realistic goals and make informed decisions.

Course and Institute Selection: AIML can assist students in making optimal choices by providing insights into which courses and institutes they are likely to secure admission based on their academic performance and exam scores.

Trend Analysis: AIML can continuously monitor and update predictions based on real-time data, helping students stay current with changing admission trends and adjust their strategies accordingly.

Student Counseling: Educational institutions can use AIML for counseling services, providing students with personalized advice and recommendations for course selection based on their academic profiles and aspirations.

Resource Optimization: For the institutes themselves, AIML can help optimize resource allocation and admission processes based on historical data and future predictions, ensuring efficient use of available seats.

AIML's role in IIT and NIT admission cutoff rank analysis can greatly benefit students, educational institutions, and policymakers by making the admission process more data-driven, transparent, and efficient. It enables students to make informed choices and institutions to make data-backed decisions, ultimately improving the overall quality of education and admissions in these prestigious institutes.

2.0 Internship Project - Data Link

Data containing information about categories, gender, courses, years, cutoff ranks, and institutes such as NIIT (National Institutes of Information Technology) and IIT (Indian Institutes of Technology) is vital for understanding and analyzing the dynamics of admissions in these prestigious institutions. This dataset offers valuable insights into the competitive landscape of education.

This predictive capability helps students, parents, and educational institutions make informed decisions. AIML also allows for personalized counseling, guiding students to choose the best courses based on their profiles and aspirations. Moreover, institutions can optimize resource allocation and admission processes, ensuring fairness and transparency. Overall, AIML enhances the efficiency and accuracy of the admiss.This predictive capability helps students, parents, and educational institutions make informed decisions. AIML also allows for personalized counseling, guiding students to choose the best courses based on their profiles and aspirations. Moreover, institutions can optimize resource allocation and admission processes, ensuring fairness and transparency. Overall,AIML enhances the efficiency and accuracy of the admission process, benefiting all stakeholders involved.ion process, benefiting all stakeholders involved.

https://www.kaggle.com/datasets/rumbleftw/iit-nit-data

3.0 AI / ML Modelling and Results

3.1 Your Problem of Statement

Predictive models are most effective when they are constructed using a company's own historical claims data since this allows the model to recognize the specific nature of a company's exposure as well as its claims practices. The construction of the model also involves input from the company throughout the process, as well as consideration of industry leading claims practices and benchmarks.

Predictive modelling can be used to quantify the impact to the claims department resulting from the failure to meet or exceed claim service leading practices. It can also be used to identify the root cause of claim leakage. Proper use of predictive modelling will allow for potential savings across two dimensions:

Early identification of claims with the potential for high leakage, thereby allowing for the proactive management of the claim

Recognition of practices that are unnecessarily increasing claims settlement payments.

3.2 Data Science Project Life Cycle

Data Science is a multidisciplinary field of study that combines programming skills, domain expertise and knowledge of statistics and mathematics to extract useful insights and knowledge from data.

In simple terms, a data science life cycle is nothing but a repetitive set of steps that you need to take to complete and deliver a project/product to your client.

Although the data science projects and the teams involved in deploying and developing the model will be different, every data science life cycle will be slightly different in every other company.
However, most of the data science projects happen to follow a somewhat similar process.
In order to start and complete a data science-based project, we need to understand the various roles and responsibilities of the people involved in building, developing the project.

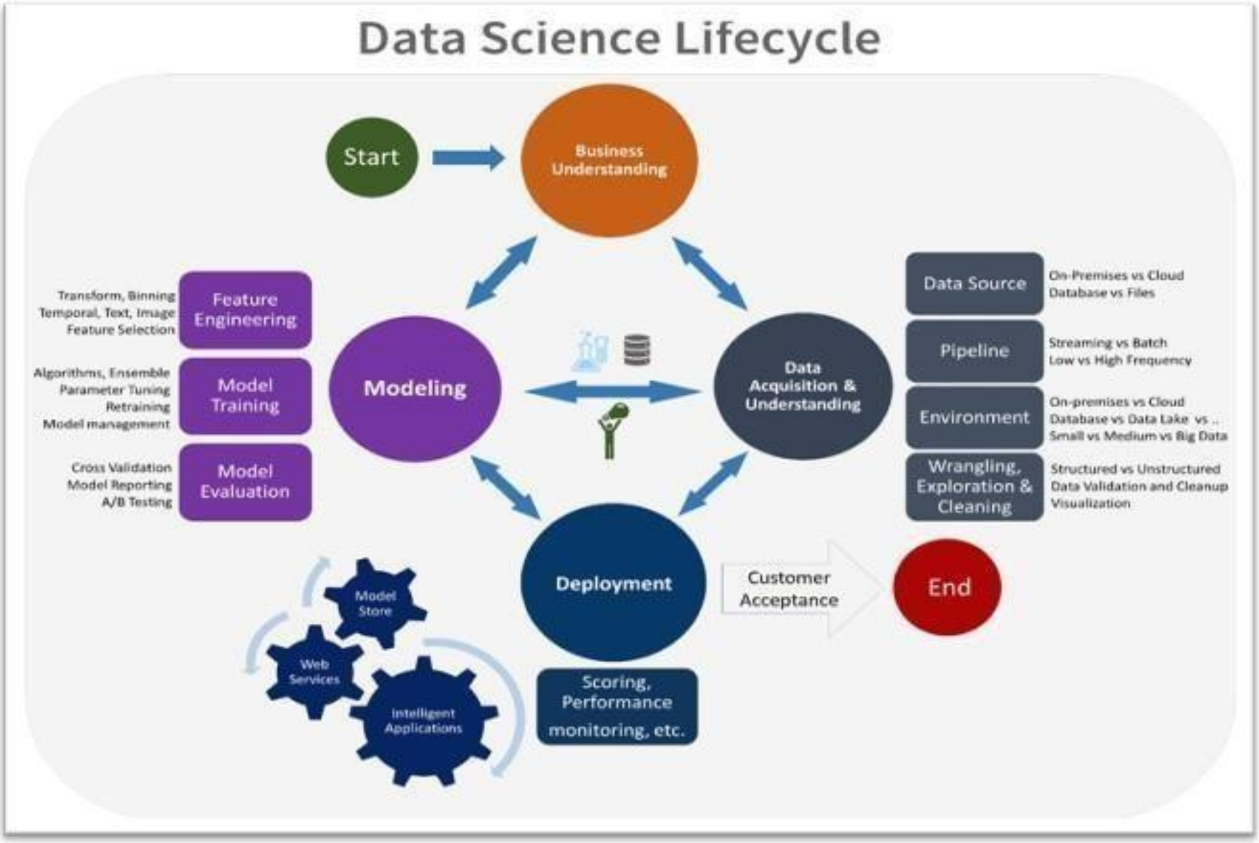
Let us look at those employees who are involved in a typical data science project:

Who Are Involved in The Projects:

○ Business Analyst ○ Data
Analyst ○ Data Scientists ○ Data

Engineer ○ Data Architect ○

Machine Learning Engineer



3.2.1 Data Exploratory Analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

3.2.2 Data Pre-processing

Data pre-processing, a component of data preparation, describes any type of processing performed on raw data to prepare it for another data processing procedure. It has traditionally been an important preliminary step for the data mining process. More recently, data preprocessing techniques have been adapted for training machine learning models and AI models and for running inferences against them.

Data pre-processing transforms the data into a format that is more easily and effectively processed in data mining, machine learning and other data science tasks. The techniques are generally used at the earliest stages of the machine learning and AI development pipeline to ensure accurate results.

3.2.1.1 Check the Duplicate and low variation data

Types of Duplicate Features in Machine Learning:

Two things distinguish top data scientists from others in most cases: Feature Creation and Feature Selection. i.e., creating features that capture deeper/hidden insights about the business or customer and then making the right choices about which features to choose for your model.

1. Duplicate Values (Same value for each record)

2. Duplicate Index (value of two features are different but they occur at the same index)

Keeping duplicate features in your dataset introduces the problem of multicollinearity.

— In the case of linear models, weights distribution between the two features will be problematic.

— If you are using tree-based modes, it won't matter unless you are looking at feature importance.

— In the case of distance-based models, it will make that feature count more in the distance.

3.2.2.2 Identify and address the missing variables

What are missing data?

Missing data are values that are not recorded in a dataset. They can be a single value missing in a single cell or missing of an entire observation (row). Missing data can occur both in a continuous variable (e.g., height of students) or a categorical variable (e.g., gender of a population).

Missing data are common in any field of natural or behavioral science, but it is particularly commonplace in social sciences research data.

In programming languages, missing values are represented as NA or Nan or simply an empty cell in an object.

• The origins of missing data

So where do the missing values come from, and why do they even exist?

Let's give an example. You are administering a questionnaire survey among a sample of respondents; and in the questionnaire, you are asking a question about household income. Now, what if a respondent refuses to answer that question? Would you make that up or rather leave the field empty? You'd probably leave that cell empty — creating an instance of missing value

• Problems caused

Missing values are undesirable, but it is difficult to quantify the magnitude of effects in statistical and machine learning projects. If it's a large dataset and a very small percentage of data is missing the effect may not be detectable at all.

However, if the dataset is relatively small, every data point counts. In these situations, a missing data point means loss of valuable information.

In any case, generally missing data creates imbalanced observations, cause biased estimates, and in extreme cases, can even lead to invalid conclusion.

Case deletion: if the dataset is relatively large delete the complete record with a missing value
Substitution: substitute missing cells with (a) column mean, (b) mean of nearest neighbors, (c) moving average, or (c) filling with the last observation

Statistical imputation: a regression can be an effective way to determine the value of missing cell given other information in the dataset

Sensitivity analysis: if the sample is small or missing values are relatively large then conduct a sensitivity analysis with multiple variations of outcomes.

3.2.2.2 Identify objects and convert into numerical values:

- ✓ **Defining data types when reading a CSV file**
- ✓ **Creating a custom function to convert data type**
- ✓ **as type () vs. to numeric ()**

When doing data analysis, it is important to ensure correct data types. Otherwise, you may get unexpected results or errors. In the case of Pandas, it will correctly infer data types in many cases, and you can move on with your analysis without any further thought on the topic.

Despite how well pandas works at some point in your data analysis process you will likely need to explicitly convert data from one type to another. This article will discuss how to change data to a numeric type. More specifically, you will learn how to use the Pandas built-in methods as type () and to numeric () to deal with the following common problems:

- ✓ Converting string/int to int/float
- ✓ Converting float to int
- ✓ Converting a column of mixed data types
- ✓ Handling missing values
- ✓ Converting a money column to float
- ✓ Converting Boolean to 0/1
- ✓ Converting multiple data columns at once

3.2.2.3 Handling of Outliers

What is an outlier?

An outlier is an observation that lies an abnormal distance from other values in a random sample from a population.

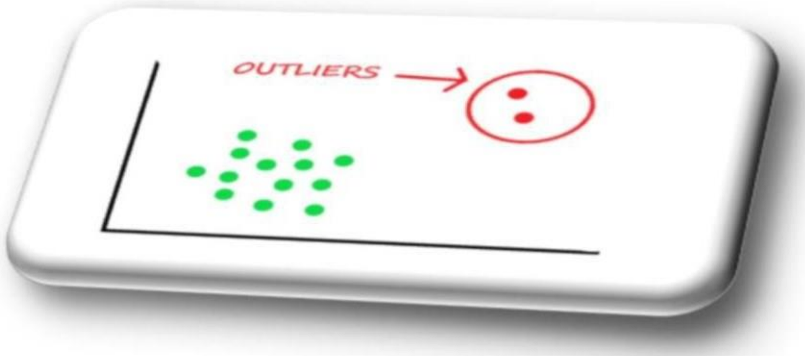
There is, of course, a degree of ambiguity. Qualifying a data point as an anomaly leaves it up to the analyst or model to determine what is abnormal—and what to do with such data points.

There are also different degrees of outliers:

- Mild outliers lie beyond an "inner fence" on either side.
- Extreme outliers are beyond an "outer fence."

Why do outliers occur? According to Tom Barenberg, chief economist and data consultant at Unity Marketing, "It can be the result of measurement or recording errors, or the unintended and truthful outcome resulting from the set's definition."

Outliers may contain valuable information. Or be meaningless aberrations caused by measurement and recording errors. In any case, they can cause problems with repeatable A/B test results, so it's important to question and analyze outliers.



3.2.2.4 Categorical data and Encoding Techniques

What is Categorical Data?

Since we are going to be working on categorical variables in this article, here is a quick refresher on the same with a couple of examples. Categorical variables are usually represented as ‘strings’ or ‘categories’ and are finite in number. Here are a few examples:

1. The city where a person lives: Delhi, Mumbai, Ahmedabad, Bangalore, etc.

2. The department a person works in: Finance, Human resources, IT, Production.

3. The highest degree a person has: High school, Diploma, Bachelors, Masters, PhD.

4. The grades of a student: A+, A, B+, B, B- etc.

In the above examples, the variables only have definite possible values. Further, we can see there are two kinds of categorical data-

- Ordinal Data: The categories have an inherent order
- Nominal Data: The categories do not have an inherent order

Label Encoding:

- We use this categorical data encoding technique when the categorical feature is ordinal. In this case, retaining the order is important. Hence encoding should reflect the sequence.
- In Label encoding, each label is converted into an integer value. We will create a variable that contains the categories representing the education qualification of a person.

Binary Encoding:

- Binary encoding is a combination of Hash encoding and one-hot encoding. In this encoding scheme, the categorical feature is first converted into numerical using an ordinal encoder. Then the numbers are transformed in the binary number. After that binary value is split into different columns.
- Binary encoding works well when there are a high number of categories. For example, the cities in a country where a company supplies its products

3.2.2.5 Feature Scaling

Why Feature Scaling?

Real Life Datasets have many features with a wide range of values like for example let's consider the house price prediction dataset. It will have many features like no. of. bedrooms, square feet area of the house, etc.

As you can guess, the no. of bedrooms will vary between 1 and 5, but the square feet area will range from 500-2000. This is a huge difference in the range of both features.

Many machine learning algorithms that are using Euclidean distance as a metric to calculate the similarities will fail to give a reasonable recognition to the smaller feature, in this case, the number of bedrooms, which in the real case can turn out to be an important metric.

E.g.: Linear Regression, Logistic Regression, KNN

There are several ways to do feature scaling. I will be discussing the top 5 of the most used feature scaling techniques.

3.2.3 Selection of Dependent and Independent variables

The dependent or target variable here is satisfaction Target which tells us a

The independent variables are selected after doing exploratory data analysis. Which tells us that the Customer is satisfied, neutral or dissatisfied.

3.2.4 Data Sampling Methods

The data we have is highly unbalanced data so we used some sampling methods which are used to balance the target variable so we our model will be developed with good accuracy and precision. We used three Sampling methods

3.2.4.1 Stratified sampling

Stratified sampling randomly selects data points from majority class so they will be equal to the data points in the minority class. So, after the sampling both the class will have same no of observations.

It can be performed using strata function from the library sampling.

3.2.4.2 Simple random sampling

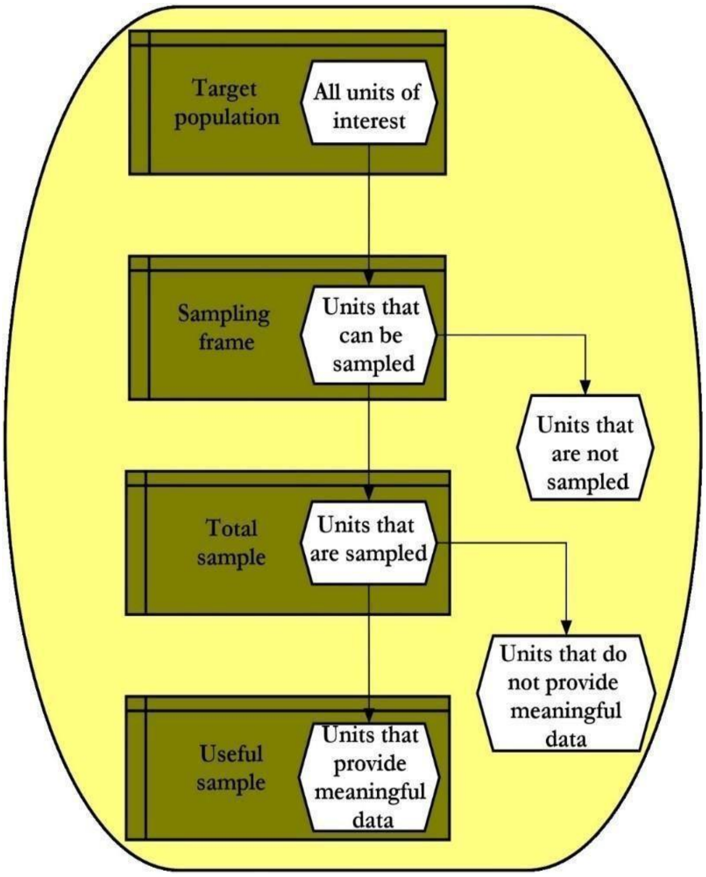
Simple random sampling is a sampling technique where a set percentage of the data is selected randomly. It is generally done to reduce bias in the dataset which can occur if data is selected manually without randomizing the dataset.

We used this method to split the dataset into train dataset which contains 70% of the total data and test dataset with the remaining 30% of the data.

Steps involved in Sampling:

The first stage in the sampling process is to clearly define the target population.

- So, to carry out opinion polls, polling agencies consider only the people who are above 18 years of age and are eligible to vote in the population. Sampling Frame – It is a list of items or people forming a population from which the sample is taken.
- So, the sampling frame would be the list of all the people whose names appear on the voter list of a constituency.
- Generally, probability sampling methods are used because every vote has equal value and any person can be included in the sample irrespective of his caste, community, or religion.
  Different samples are taken from different regions all over the country.
- Sample Size – It is the number of individuals or items to be taken in a sample that would be enough to make inferences about the population with the desired level of accuracy and precision
- Once the target population, sampling frame, sampling technique, and sample **size** have been established, the next step is to collect data from the sample.



3.2.5 Models Used for Development

3.2.5.1 Model 01(Logistic regression)

Logistic uses logistic link function to convert the likelihood values to probabilities so we can get a good estimate on the probability of a particular observation to be positive class or negative class. The also gives us p-value of the variables which tells us about significance of each independent variable.

3.2.5.2 Model 02(Decision Tree Classifier)

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a treestructured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed based on features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

It is called a decision tree because, like a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features. A tree can be seen as a piece wise constant approximation

3.2.5.3 Model 03(Random Forest Classifier)

Random forest is an algorithm that consists of many decision trees. It was first developed by Leo Bierman and Adele Cutler. The idea behind it is to build several trees, to have the instance classified by each tree, andto give a "vote" at each class. The model uses a "bagging" approach and the random selection of features to build a collection of decision trees with controlled variance. The instance's class is to the class with the highest number of votes, the class that occurs the most within the leaf in which the instance is placed.

The error of the forest depends on:

- Trees correlation: the higher the correlation, the higher the forest error rate.
- The strength of each tree in the forest. A strong tree is a tree with low error. By using trees that classify the instances with low error the error rate of the forest decreases.

3.2.5.4 Model 04(Extra Trees Classifier)

This class implements a meta estimator that fits several randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

3.2.5.5 Model 05(KNN Classifier)

K-Nearest Neighbor is one of the simplest Machine Learning algorithms based on Supervised Learning technique K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most like the available categories'-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using KNN algorithm.

K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset KNN-algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much like the new data.

3.2.5.6 Model 06(Naïve Bayes)

Naïve Bayes is a probabilistic machine learning algorithm used for many classification functions and is based on the Bayes theorem. Gaussian Naïve Bayes is the extension of naïve Bayes. While other functions are used to estimate data distribution, Gaussian or normal distribution is the simplest to implement as you will need to calculate the mean and standard deviation for the training data.

What is the Naive Bayes Algorithm?

Naive Bayes is a probabilistic machine learning algorithm that can be used in several classification tasks. Typical applications of Naive Bayes are classification of documents, filtering spam, prediction and so on. This algorithm is based on the discoveries of Thomas Bayes and hence its name.

The name “Naïve” is used because the algorithm incorporates features in its model that are independent of each other. Any modifications in the value of one feature do not directly impact the value of any other feature of the algorithm. The main advantage of the Naïve Bayes algorithm is that it is a simple yet powerful algorithm.

It is based on the probabilistic model where the algorithm can be coded easily, and predictions did quickly in real-time. Hence this algorithm is the typical choice to solve realworld problems as it can be tuned to respond to user requests instantly. But before we dive deep into Naïve Bayes and Gaussian Naïve Bayes, we must know what is meant by conditional probability.

3.2.5.7 Model 07(XG Boost)

XG Boost is an implementation of Gradient Boosted decision trees. This library was written in C++. It is a type of Software library that was designed basically to improve speed and model performance. It has recently been dominating in applied machine learning. XG Boost models majorly dominate in many Kaggle

Competitions. In this algorithm, decision trees are created in sequential form. Weights play an important role in XG Boost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

3.2.5.8 Model 08(Light GBM)

Light GBM is a gradient boosting framework based on decision trees to increases the efficiency of the model and reduces memory usage.

It uses two novel techniques: Gradient-based One Side Sampling and Exclusive Feature Bundling (EFB) which fulfills the limitations of histogram-based algorithm that is primarily used in all GBDT (Gradient Boosting Decision Tree) frameworks. The two techniques of GOSS and EFB described below form the characteristics of Light GBM Algorithm. They comprise together to make the model work efficiently and provide it a cutting edge over other GBDT frameworks.

Gradient-based One Side Sampling Technique for Light GBM:

Different data instances have varied roles in the computation of information gain. The instances with larger gradients (i.e., under-trained instances) will contribute more to the information gain. GOSS keeps those instances with large gradients (e.g., larger than a predefined threshold, or among the top percentiles), and only randomly drop those instances with small gradients to retain the accuracy of information gain estimation. This treatment can lead to a more accurate gain estimation than uniformly random sampling, with the same target sampling rate, especially when the value of information gain has a large range.

3.2.5.9 Model 09 (SVC)

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate ndimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

3.3 AI / ML Model Analysis and Final Results

We used our train dataset to build the above models and used our test data to check the accuracy and performance of our models.

We used confusion matrix to check accuracy, Precision, Recall and F1 score of our models and compare and select the best model for given auto dataset of size ~ 272252 policies.

3.3.1 Different Model codes

This section in which we used different types of model code as follows :

```
# 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
```

In [2]:

```
# Load the dataset information

data = pd.read_excel(r"data(New).xlsx", header=0)

# Copy to back-up file

data_bk =data.copy()

# Display first 5 records

data.head()
```

Out[2]:

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory
0	1	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	GEN	838	1841	0
1	2	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	OBC-NCL	408	1098	0
2	3	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	SC	297	468	0
3	4	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	ST	79	145	0
4	5	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	GEN-PWD	94	94	0

In [3]:

```
# Display the dataset information

data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 64958 entries, 0 to 64957
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                64958 non-null  int64
 1   year              64958 non-null  int64
 2   institute_type    64958 non-null  object
 3   round_no          64958 non-null  int64
 4   quota             64958 non-null  object
 5   pool              64958 non-null  object
 6   institute_short   64958 non-null  object
 7   program_name      64958 non-null  object
 8   program_duration  64958 non-null  object
 9   degree_short      64958 non-null  object
 10  category          64958 non-null  object
 11  opening_rank      64958 non-null  int64
 12  closing_rank      64958 non-null  int64
 13  is_preparatory    64958 non-null  int64
dtypes: int64(6), object(8)
memory usage: 6.9+ MB
```

In [4]:

```
# display the unique values of the all the variables
data.nunique()
```

Out[4]:

```
id                   25458
year                     6
```

```
institute_type          2
round_no                4
quota                   7
pool                    2
institute_short        54
program_name          130
program_duration        2
degree_short           13
category               10
opening_rank        10984
closing_rank        11940
is_preparatory          2
dtype: int64
```

In [5]:

```
# display the shape of the dataset
data.shape
```

Out[5]:

```
(64958, 14)
```

In [6]:

```
# display the duplicated values with in  dataset
data.duplicated().any()
```

Out[6]:

```
True
```

In [7]:

```
# display duplicate values with in dataset
data_dup=data[data.duplicated(keep='last')]
# disply the duplicate records
data_dup
```

Out[7]:

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory
9205	9206	2021	IIT	1	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	GEN	123	2003	0
9206	9207	2021	IIT	1	AI	Female-Only	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	GEN	702	4419	0
9207	9208	2021	IIT	1	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	OBC-NCL	389	1123	0
9208	9209	2021	IIT	1	AI	Female-Only	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	OBC-NCL	1618	2505	0
9209	9210	2021	IIT	1	AI	Gender-Neutral	IIT-Bombay	Aerospace Engineering	4 Years	B.Tech	SC	129	579	0

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
64903	31136	2021	NIT	1	JK	Female-Only	NIT-Srinagar	Electronics and Communication Engineering	4 Years	B.Tech	SC	14185	24048	0
64904	31137	2021	NIT	1	JK	Gender-Neutral	NIT-Srinagar	Electronics and Communication Engineering	4 Years	B.Tech	ST	2736	4171	0
64905	31138	2021	NIT	1	JK	Female-Only	NIT-Srinagar	Electronics and Communication Engineering	4 Years	B.Tech	ST	10870	10870	0
64906	31139	2021	NIT	1	LA	Gender-Neutral	NIT-Srinagar	Electronics and Communication Engineering	4 Years	B.Tech	GEN	166453	265454	0
64907	31140	2021	NIT	1	LA	Female-Only	NIT-Srinagar	Electronics and Communication Engineering	4 Years	B.Tech	GEN	215054	215054	0

39500 rows × 14 columns

In [8]:

```
# remove the identified duplicate records
data=data.drop_duplicates()

# display the shape of the dataset
data.shape
```

Out[8]:

```
(25458, 14)
```

In [9]:

```
# Re-setting the raw index
data=data.reset_index(drop=True)
# copy file to backup file after deletion of duplicate records
data_bk2=data.copy()
```

In [10]:

```
# display the duplicated  values in the dataset
data.duplicated().any()
```

Out[10]:

```
False
```

In [11]:

```
# display the missing values information of variables
data.isnull().sum()
```

Out[11]:

```
id                   0
year                 0
institute_type       0
round_no             0
quota                0
pool                 0
institute_short      0
program_name         0
program_duration     0
degree_short         0
category             0
opening_rank         0
closing_rank         0
is_preparatory       0
dtype: int64
```

In [12]:

```
# display the descriptive status
data.describe()
```

Out[12]:

	id	year	round_no	opening_rank	closing_rank	is_preparatory
count	25458.000000	25458.000000	25458.000000	2.545800e+04	2.545800e+04	25458.000000
mean	15065.188978	2019.524118	4.864993	8.347711e+03	1.100359e+04	0.035706
std	9630.192936	1.431272	2.530553	2.946525e+04	4.170573e+04	0.185559
min	1.000000	2016.000000	1.000000	0.000000e+00	0.000000e+00	0.000000
25%	6365.250000	2019.000000	1.000000	6.550000e+02	8.260000e+02	0.000000
50%	12729.500000	2020.000000	6.000000	2.237000e+03	2.715000e+03	0.000000
75%	23803.750000	2021.000000	7.000000	6.781750e+03	8.155500e+03	0.000000
max	31140.000000	2021.000000	7.000000	1.082601e+06	1.144790e+06	1.000000

In [77]:

```
# to find outliers
first_quantile=data['id'].quantile(.25)
third_quantile=data['id'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.id < lower_bound) | (data.id > upper_bound)]
```

Out[77]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [78]:

```
# to find outliers
first_quantile=data['year'].quantile(.25)
third_quantile=data['year'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.year < lower_bound) | (data.year > upper_bound)]
```

Out[78]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [79]:

```
# to find outliers
first_quantile=data['institute_type'].quantile(.25)
third_quantile=data['institute_type'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.institute_type < lower_bound) | (data.institute_type > upper_bound)]
```

Out[79]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [80]:

```
# to find outliers
first_quantile=data['round_no'].quantile(.25)
third_quantile=data['round_no'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.round_no < lower_bound) | (data.round_no > upper_bound)]
```

Out[80]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [81]:

```
# to find outliers
first_quantile=data['quota'].quantile(.25)
third_quantile=data['quota'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.quota  < lower_bound) | (data.quota  > upper_bound)]
```

Out[81]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [82]:

```
# to find outliers
first_quantile=data['pool'].quantile(.25)
third_quantile=data['pool'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.pool < lower_bound) | (data.pool  > upper_bound)]
```

Out[82]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [83]:

```
# to find outliers
first_quantile=data['institute_short'].quantile(.25)
third_quantile=data['institute_short'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.institute_short < lower_bound) | (data.institute_short  > upper_bound)]
```

Out[83]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [84]:

```
# to find outliers
first_quantile=data['program_name'].quantile(.25)
third_quantile=data['program_name'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.program_name < lower_bound) | (data.program_name  > upper_bound)]
```

Out[84]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [85]:

```
# to find outliers
first_quantile=data['category'].quantile(.25)
third_quantile=data['category'].quantile(.75)
IQR=third_quantile-first_quantile
upper_bound=round(third_quantile+1.5*IQR,3)
upper_bound
lower_bound=round(first_quantile-1.5*IQR,3)
lower_bound
data[(data.category < lower_bound) | (data.category  > upper_bound)]
```

Out[85]:

id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory

In [13]:

```
# display the institute_type variables count
data['institute_type'].value_counts()
```

Out[13]:

```
IIT    13155
NIT    12303
Name: institute_type, dtype: int64
```

In [14]:

```
# replace the 'institute_type' varaible and covert to integer value
data['institute_type']=data['institute_type'].str.replace('IIT','1')
data['institute_type']=data['institute_type'].str.replace('NIT','0')
data['institute_type']=data['institute_type'].astype(int)
```

In [15]:

```
# display the institute_type variables count
data['institute_type'].value_counts()
```

Out[15]:

```
1    13155
0    12303
Name: institute_type, dtype: int64
```

In [16]:

```
# display the pool variables count
data['pool'].value_counts()
```

Out[16]:

```
Gender-Neutral    16005
Female-Only        9453
Name: pool, dtype: int64
```

In [17]:

```
# replace the 'pool' varaible and covert to integer value
data['pool']=data['pool'].str.replace('Gender-Neutral','1')
data['pool']=data['pool'].str.replace('Female-Only','0')
data['pool']=data['pool'].astype(int)
```

In [18]:

```
# display the pool variables count
data['pool'].value_counts()
```

Out[18]:

```
1    16005
0     9453
Name: pool, dtype: int64
```

In [19]:

```
# display the program_duration variables count
data['program_duration'].value_counts()
```

Out[19]:

```
4 Years    21104
5 Years     4354
Name: program_duration, dtype: int64
```

In [20]:

```
# replace the 'program_duration' varaible and covert to integer value
data['program_duration']=data['program_duration'].str.replace('4 Years','1')
data['program_duration']=data['program_duration'].str.replace('5 Years','0')
data['program_duration']=data['program_duration'].astype(int)
```

In [21]:

```
# display the program_duration variables count
data['program_duration'].value_counts()
```

Out[21]:

```
1    21104
0     4354
```

```
Name: program_duration, dtype: int64
```

In [22]:

```
# display the quota  variables count
data['quota'].value_counts()
```

Out[22]:

```
AI    13155
OS     6502
HS     5486
JK      128
GO       95
AP       72
LA       20
Name: quota, dtype: int64
```

In [23]:

```
#  use the labelEncoder to handle categorical data
from sklearn.preprocessing import LabelEncoder
LE= LabelEncoder()
data['quota']=LE.fit_transform(data[['quota']])
```

In [24]:

```
# display the institute_short  variables count
data['institute_short'].value_counts()
```

Out[24]:

```
IIT-Kharagpur              2120
IIT-(BHU) Varanasi         1088
NIT-Rourkela               1054
IIT-Bombay                 1034
IIT-Delhi                  1018
IIT-Roorkee                 989
IIT-Madras                  949
IIT-Kanpur                  844
NIT-Raipur                  748
IIT-(ISM) Dhanbad           739
NIT-Calicut                 695
NIT-Hamirpur                686
NIT-Jalandhar               649
NIT-Karnataka-Surathkal     632
NIT-Bhopal                  624
NIT-Durgapur                605
NIT-Allahabad               605
IIT-Bhubaneswar             582
NIT-Agartala                561
NIT-Jaipur                  549
IIT-Guwahati                531
IIT-Hyderabad               484
NIT-Kurukshetra             477
NIT-Patna                   468
NIT-Jamshedpur              460
NIT-Srinagar                439
NIT-Silchar                 409
IIT-Ropar                   311
NIT-Warangal                306
NIT-Tiruchirappalli         303
IIT-Patna                   290
IIT-Mandi                   275
IIT-Gandhinagar             272
IIT-Jodhpur                 265
NIT-Goa                     264
IIT-Indore                  244
IIT-Jammu                   240
NIT-Puducherry              238
IIT-Tirupati                230
```

```
IIT-Palakkad                190
NIT-Arunachal-Pradesh       188
NIT-Manipur                 188
NIT-Meghalaya               175
NIT-Delhi                   162
IIT-Goa                     161
NIT-Nagaland                160
IIT-Bhilai                  155
NIT-Mizoram                 153
NIT-Sikkim                  147
IIT-Dharwad                 144
NIT-Uttarakhand              99
NIT-Surat                    93
NIT-Nagpur                   92
NIT-Andhra-Pradesh           74
Name: institute_short, dtype: int64
```

In [25]:

```python
#  use the labelEncoder to handle categorical data
from sklearn.preprocessing import LabelEncoder
LE= LabelEncoder()
data['institute_short']=LE.fit_transform(data[['institute_short']])
```

In [26]:

```python
# display the program_name variables count
data['program_name'].value_counts()
```

Out[26]:

```
Computer Science and Engineering
            3330
Mechanical Engineering
            2774
Civil Engineering
            2566
Electrical Engineering
            2279
Electronics and Communication Engineering
            1869

            ...
Manufacturing Science and Engineering with M.Tech. in Industrial andSystems Engineering and Ma
nagement       7
Industrial and Systems Engineering with M.Tech. in Industrial and SystemsEngineering and Manag
ement          7
Agricultural and Food Engineering with M.Tech. in any of the listedspecializations
               7
Engineering Physics and M.Tech. with specialization in Nano Science
               5
Civil Engineering with M.Tech. in Structural Engineering
               4
Name: program_name, Length: 130, dtype: int64
```

In [27]:

```python
#  use the labelEncoder to handle categorical data
from sklearn.preprocessing import LabelEncoder
LE= LabelEncoder()
data['program_name']=LE.fit_transform(data[['program_name']])
```

In [28]:

```python
# display the degree_short variables count
data['degree_short'].value_counts()
```

Out[28]:

```
B.Tech                   20456
B.Tech + M.Tech (IDD)     2560
BSc                        590
B.Arch                     538
```

```
Int MSc.                    298
Btech + M.Tech (IDD)        293
Int M.Tech                  249
Int Msc.                    233
BS + MS (IDD)               110
BSc + MSc (IDD)              69
B.Plan                       54
B.Pharm                       4
B.Pharm + M.Pharm             4
Name: degree_short, dtype: int64
```

In [29]:

```
#  use the labelEncoder to handle categorical data
from sklearn.preprocessing import LabelEncoder
LE= LabelEncoder()
data['degree_short']=LE.fit_transform(data[['degree_short']])
```

In [30]:

```
# display the category variables count
data['category'].value_counts()
```

Out[30]:

```
GEN            5252
OBC-NCL        4986
SC             4908
ST             4327
GEN-EWS        3205
GEN-PWD        1565
OBC-NCL-PWD     770
GEN-EWS-PWD     185
SC-PWD          182
ST-PWD           78
Name: category, dtype: int64
```

In [31]:

```
#  use the labelEncoder to handle categorical data
from sklearn.preprocessing import LabelEncoder
LE= LabelEncoder()
data['category']=LE.fit_transform(data[['category']])
```

In [32]:

```
# Display the dataset information

data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25458 entries, 0 to 25457
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 25458 non-null  int64
 1   year               25458 non-null  int64
 2   institute_type     25458 non-null  int32
 3   round_no           25458 non-null  int64
 4   quota              25458 non-null  int32
 5   pool               25458 non-null  int32
 6   institute_short    25458 non-null  int32
 7   program_name       25458 non-null  int32
 8   program_duration   25458 non-null  int32
 9   degree_short       25458 non-null  int32
 10  category           25458 non-null  int32
 11  opening_rank       25458 non-null  int64
 12  closing_rank       25458 non-null  int64
 13  is_preparatory     25458 non-null  int64
dtypes: int32(8), int64(6)
memory usage: 1.9 MB
```

In [33]:

```
# display the sample sataset
data.sample(5)
```

Out[33]:

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory
15984	20167	2019	0	7	6	0	45	12	1	4	0	29472	41386	0
5439	5440	2019	1	7	0	1	9	98	1	4	8	352	402	0
13985	18168	2019	0	7	6	1	33	7	0	0	6	170	586	0
23023	28704	2021	0	1	3	0	30	98	1	4	6	5222	6521	0
6090	6089	2019	1	7	0	0	17	47	1	4	4	4757	5309	0

In [34]:

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

is_preparatory_count = data.is_preparatory.value_counts()
print('Class 0:', is_preparatory_count[0])
print('Class 1:', is_preparatory_count[1])
print('Proportion:', round(is_preparatory_count[0] / is_preparatory_count[1], 2), ': 1')
print('Total IIT-NIT Data records:', len(data))
Class 0: 24549
Class 1: 909
Proportion: 27.01 : 1
Total IIT-NIT Data records: 25458
```

In [35]:

```
# Identify the independent and target (dependent) variables

Indepvar=[]
for col in data.columns:
    if col != 'is_preparatory':
        Indepvar.append(col)

TargetVar='is_preparatory'

x=data[Indepvar]

y=data[TargetVar]
```

In [36]:

```
# Random oversampling can be implemented using the RandomOverSampler class

from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy=0.125)

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

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

```
(27617, 13)
(27617,)
```

In [38]:

```
# Random oversampling can be implemented using the RandomOverSampler class

from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(sampling_strategy=0.125)

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

print(x_over.shape)
print(y_over.shape)
(27617, 13)
(27617,)
```

In [39]:

```
# split the data into train and test (random sampling)

#70% data train and 30% data test

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)

# display yhe shape for train and test data

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

Out[39]:

```
((17820, 13), (7638, 13), (17820,), (7638,))
```

In [40]:

```
# Scaling the features by using MinMaxScaler

from sklearn.preprocessing import MinMaxScaler

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

x_train= mmscaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train)

x_test = mmscaler.fit_transform(x_test)
x_test = pd.DataFrame(x_test)
```

Logestic Regression Algorithm(Classifier)

In [41]:

```
# To build the 'Logistic Regression' model with random sampling
from sklearn.linear_model import LogisticRegression
# create an object for model
ModelLR= LogisticRegression()
# train the model
ModelLR.fit(x_train,y_train)
# predict themodel with test the dataset
y_pred=ModelLR.predict(x_test)
y_pred_prob=ModelLR.predict_proba(x_test)
```

In [42]:

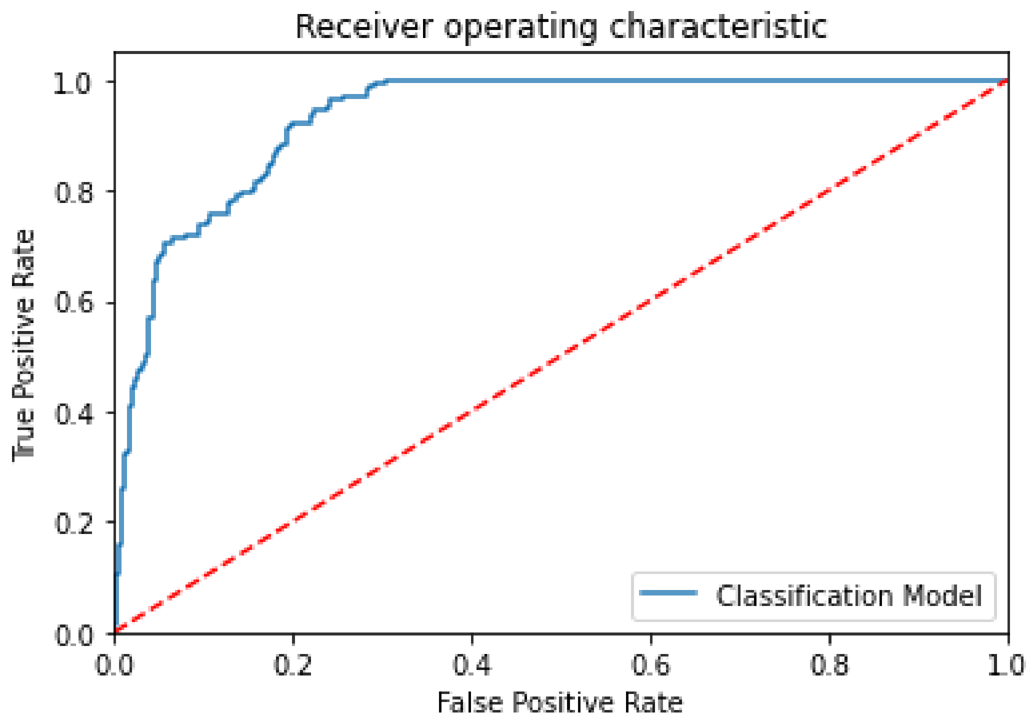
```
#To display the algorithm paramaters
params=ModelLR.get_params()
print(params)
```

```
{'C': 1.0, 'class_weight': None, 'dual': False, 'fit_intercept': True, 'intercept_scaling': 1,
 'l1_ratio': None, 'max_iter': 100, 'multi_class': 'auto', 'n_jobs': None, 'penalty': 'l2', 'r
andom_state': None, 'solver': 'lbfgs', 'tol': 0.0001, 'verbose': 0, 'warm_start': False}
```

In [43]:

```python
# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelLR.predict_proba(x_test)[:,1])
plt.figure()
#-----------------------------------------------------
plt.plot(fpr, tpr, label= 'Classification Model' % model_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.show()
print('-----------------------------------------------------------------------------------------------
--------------')
Confusion matrix :
 [[   0  262]
 [   0 7376]]
Outcome values :
 0 262 0 7376
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.48      0.50      0.49      7638
weighted avg       0.93      0.97      0.95      7638

Accuracy : 96.6 %
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
```



Decision Tree Algorithm(Classifier)

In [44]:

```
# To build the 'Decision tree algorithm' model with random sampling
from sklearn.tree import DecisionTreeClassifier
# create an object for model
ModelDT=DecisionTreeClassifier()
# train the model
ModelDT.fit(x_train,y_train)
# predict themodel with test the dataset
y_pred=ModelDT.predict(x_test)
y_pred_prob=ModelDT.predict_proba(x_test)
```

In [45]:

```
#To display the algorithm paramaters
params=ModelDT.get_params()
print(params)
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features
': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_sam
ples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': None, 'splitter': 'best'}
```

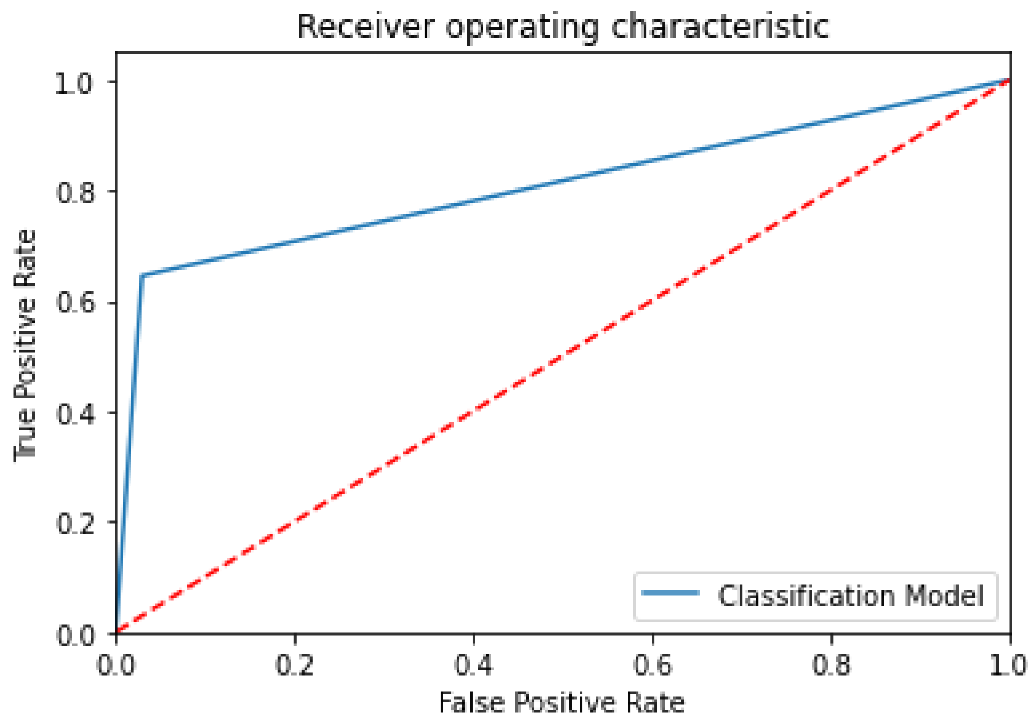
In [46]:

```
# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelDT.predict_proba(x_test)[:,1])
plt.figure()
#-----------------------------------------------------
plt.plot(fpr, tpr, label= 'Classification Model' % model_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.show()
print('----------------------------------------------------------------------------------------------
--------------')
```

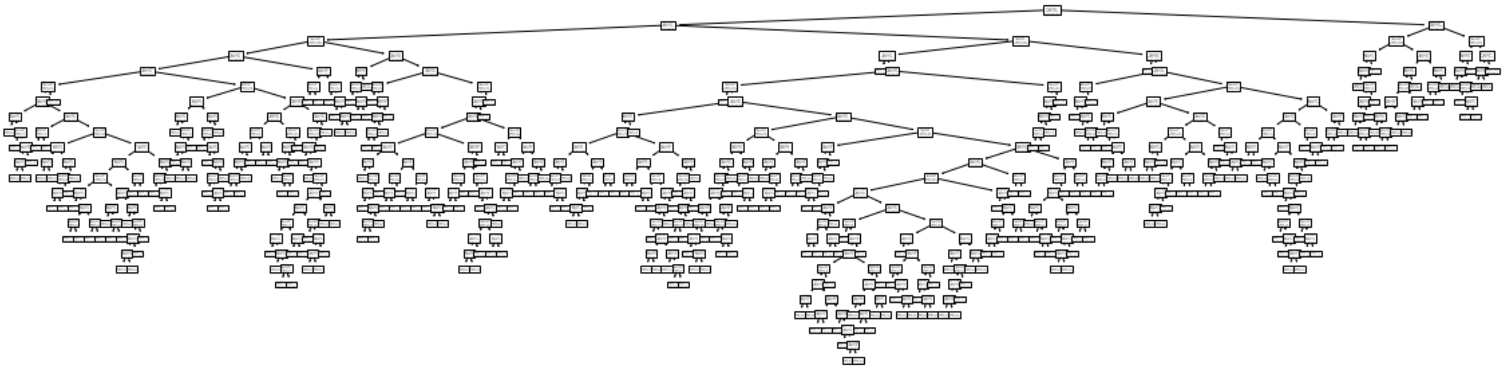
```
Confusion matrix :
 [[ 169   93]
 [ 218 7158]]
Outcome values :
 169 93 218 7158
Classification report :
               precision    recall  f1-score   support

           1       0.44      0.65      0.52       262
           0       0.99      0.97      0.98      7376

    accuracy                           0.96      7638
   macro avg       0.71      0.81      0.75      7638
weighted avg       0.97      0.96      0.96      7638

Accuracy : 95.9 %
Precision : 43.7 %
Recall : 64.5 %
F1 Score : 0.521
Specificity or True Negative Rate : 97.0 %
Balanced Accuracy : 80.8 %
MCC : 0.511
roc_auc_score: 0.808
```




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

In [47]:

```
# plot the decision tree

import matplotlib.pyplot as plt
from sklearn import tree

plt.figure(figsize=(20,5))
tree.plot_tree(ModelDT);
```



Random Forest Algorithm(Classifier)

In [48]:

```
# To build the 'Random forest algorithm' model with random sampling
from sklearn.ensemble import RandomForestClassifier
# create an object for model
ModelRF= RandomForestClassifier()
# train the model
ModelRF.fit(x_train,y_train)
# predict themodel with test the dataset
y_pred=ModelRF.predict(x_test)
y_pred_prob=ModelRF.predict_proba(x_test)
```

In [49]:

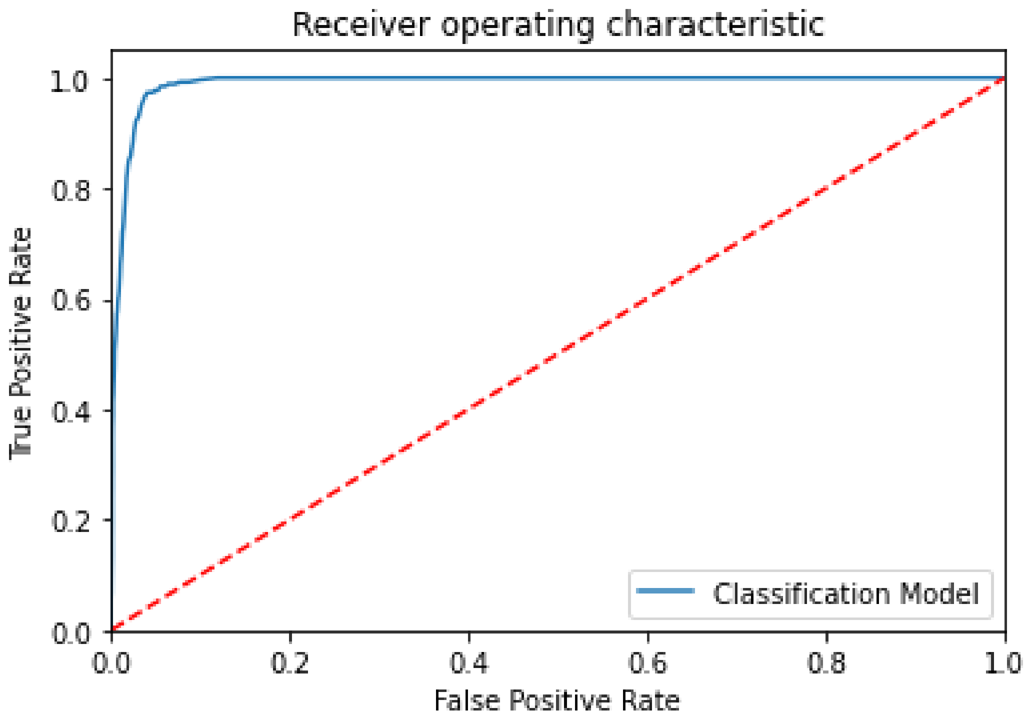
```
#To display the algorithm paramaters
params=ModelRF.get_params()
print(params)
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}
```

In [50]:

```
# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelRF.predict_proba(x_test)[:,1])
plt.figure()
#----------------------------------------------------
plt.plot(fpr, tpr, label= 'Classification Model' % model_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.show()
print('-----------------------------------------------------------------------------------------
--------------')
Confusion matrix :
 [[ 145  117]
 [  43 7333]]
Outcome values :
 145 117 43 7333
Classification report :
               precision    recall  f1-score   support
```

```
           1       0.77      0.55      0.64       262
           0       0.98      0.99      0.99      7376

    accuracy                           0.98      7638
   macro avg       0.88      0.77      0.82      7638
weighted avg       0.98      0.98      0.98      7638

Accuracy : 97.9 %
Precision : 77.1 %
Recall : 55.3 %
F1 Score : 0.644
Specificity or True Negative Rate : 99.4 %
Balanced Accuracy : 77.4 %
MCC : 0.643
roc_auc_score: 0.774
```




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

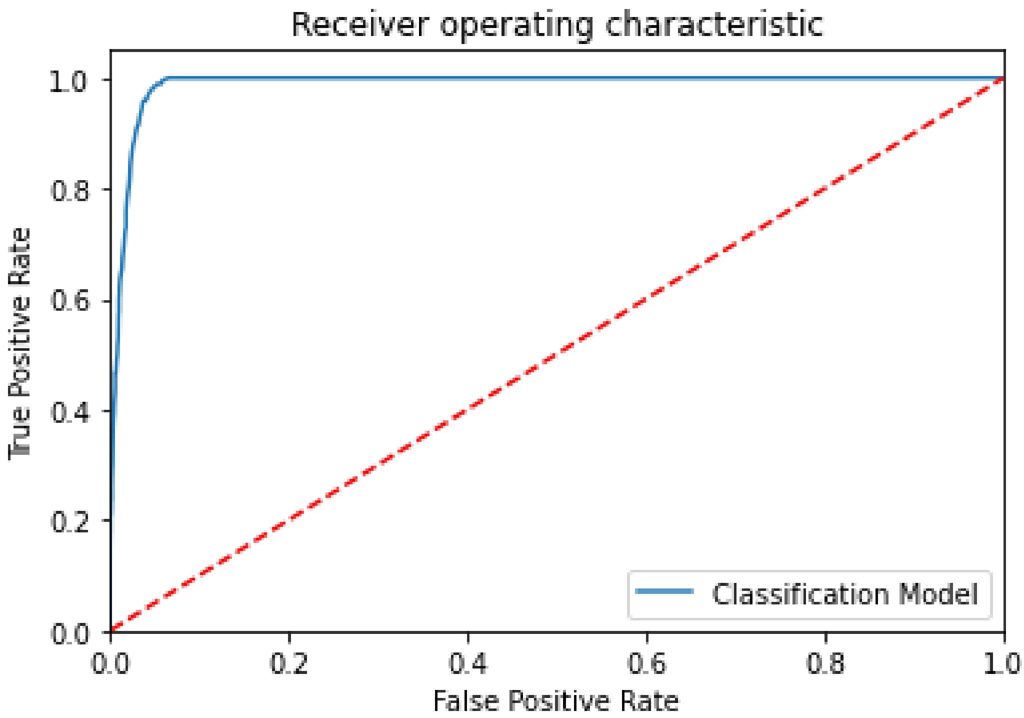
Extra Tree Algorithm(Classifier)

In [51]:

```
# To build the 'Random Forest' model with random sampling

from sklearn.ensemble import ExtraTreesClassifier

# Create an object for Extra Trees Classifier

ModelET = ExtraTreesClassifier()

# Train the model with train data

ModelET.fit(x_train,y_train)

# Predict the model with test data set

y_pred = ModelET.predict(x_test)
y_pred_prob = ModelET.predict_proba(x_test)
```

```
# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelET.predict_proba(x_test)[:,1])
plt.figure()
#-------------------------------------------------------------------
plt.plot(fpr, tpr, label= 'Classification Model' % model_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('-----------------------------------------------------------------------------------------
--------------')
Confusion matrix :
 [[ 122  140]
 [  47 7329]]
Outcome values :
 122 140 47 7329
Classification report :
               precision    recall  f1-score   support

           1       0.72      0.47      0.57       262
           0       0.98      0.99      0.99      7376

    accuracy                           0.98      7638
   macro avg       0.85      0.73      0.78      7638
weighted avg       0.97      0.98      0.97      7638

Accuracy : 97.6 %
Precision : 72.2 %
Recall : 46.6 %
F1 Score : 0.566
Specificity or True Negative Rate : 99.4 %
Balanced Accuracy : 73.0 %
MCC : 0.568
roc_auc_score: 0.73
```




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

In [52]:

```
# display the all the variables
data.columns
```

Out[52]:

```
Index(['id', 'year', 'institute_type', 'round_no', 'quota', 'pool',
       'institute_short', 'program_name', 'program_duration', 'degree_short',
       'category', 'opening_rank', 'closing_rank', 'is_preparatory'],
      dtype='object')
```

In [54]:

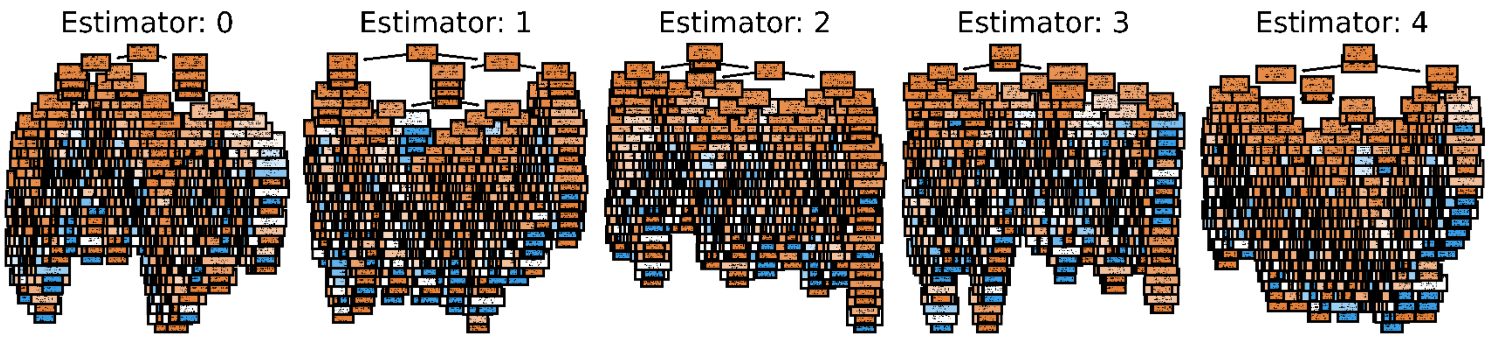
```
# Create a list for plotting the decision trees

figcols = ['id', 'year', 'institute_type', 'round_no', 'quota', 'pool',
       'institute_short', 'program_name', 'program_duration', 'degree_short',
       'category', 'opening_rank', 'closing_rank', 'is_preparatory']
```

In [55]:

```
# Visualize individual trees and code below visualizes the first decision tree of Extra Trees
Classifier

from sklearn import tree

fn1=figcols
cn1=['0', '1']

fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (4,4), dpi=800)
tree.plot_tree(ModelET.estimators_[0],
               feature_names = fn1,
               class_names=cn1,
               filled = True);
#fig.savefig('ModelET.png')
```

In [56]:

```
# Visualize individual trees and code below visualizes the first 5 decision trees of Extra
Trees Classifier

from sklearn import tree

fn2=figcols
cn2=['0', '1']

fig, axes = plt.subplots(nrows = 1, ncols = 5, figsize = (10,2), dpi=3000)
for index in range(0, 5):
    tree.plot_tree(ModelET.estimators_[index],
                   feature_names = fn2,
                   class_names=cn2,
                   filled = True,
                   ax = axes[index]);

    axes[index].set_title('Estimator: ' + str(index), fontsize = 11)
#fig.savefig('ModelET1.png')
```



KNN Algorithm

In [57]:

```
# load the KNNResults
KNNResults = pd.read_excel(r"KNN_ResultsNew.xlsx", header=0)
KNNResults.head()
```

Out[57]:

Model Name	KNN K Value	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy

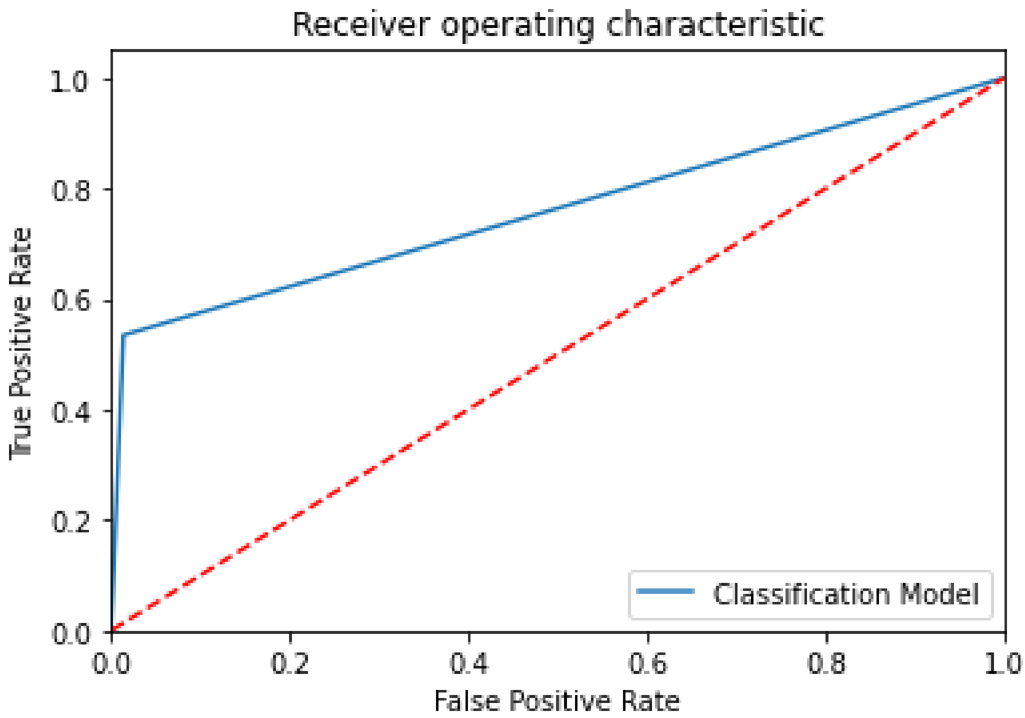
In [58]:

```
# Build KNN Model

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

import sklearn.metrics as metrics

from sklearn.metrics import roc_curve, roc_auc_score

accuracy = []

for a in range(1, 21, 1):

    k = a

    # Build the model

    ModelKNN = KNeighborsClassifier(n_neighbors=k)

    # Train the model

    ModelKNN.fit(x_train, y_train)

    # Predict the model

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

    print('KNN_K_value = ', a)

    # Print the model name

    print('Model Name: ', ModelKNN)

    # 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
```

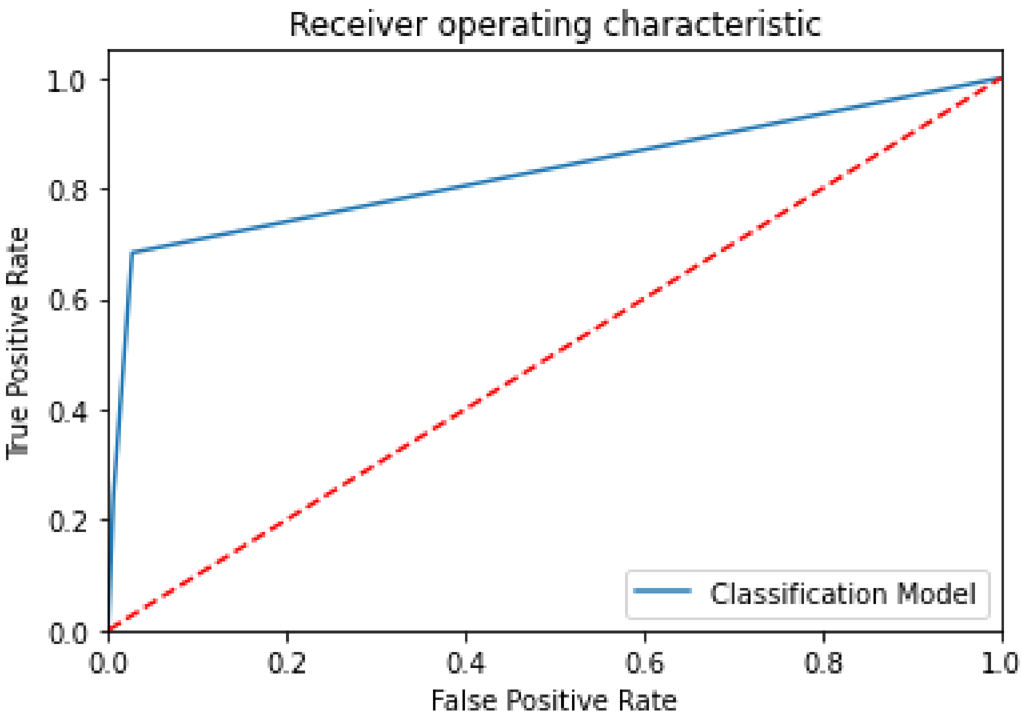
```python
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print('Outcome values : \n', tp, fn, fp, tn)

# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual, ModelKNN.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' % model_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()
#---------------------------------------------------------------------------------
new_row = {'Model Name' : ModelKNN,
```

```
                'KNN K Value' : a,
                '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}
    KNNResults = KNNResults.append(new_row, ignore_index=True)
    #------KNN_Results-----------------------------------------------------------------------
```

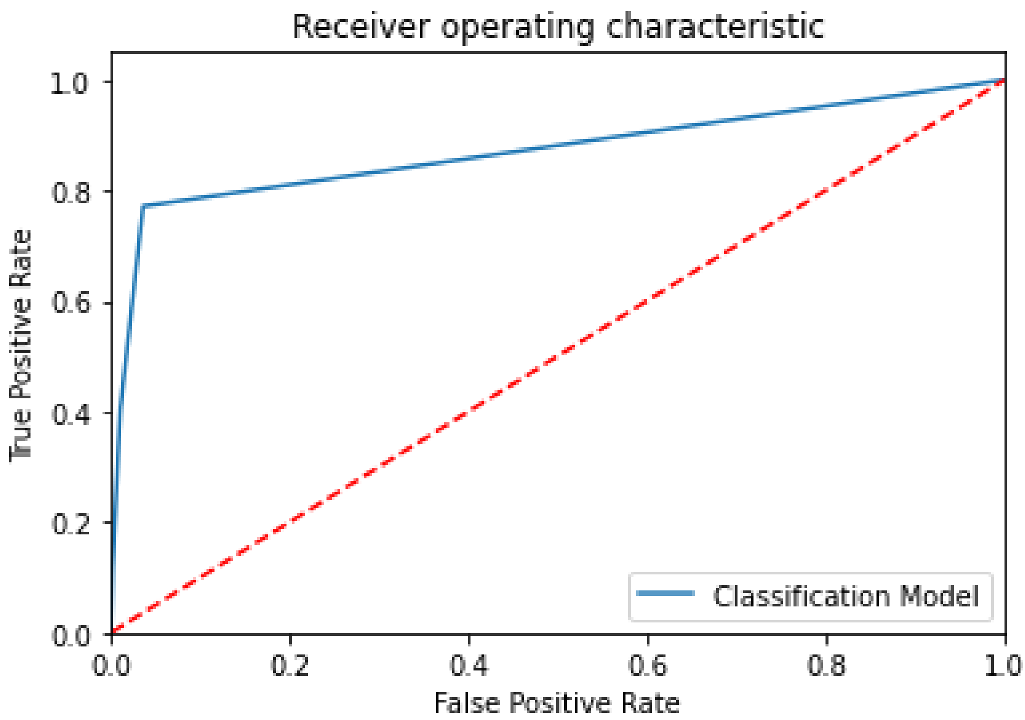
```
KNN_K_value =  1
Model Name:  KNeighborsClassifier(n_neighbors=1)
Confusion matrix :
 [[ 140  122]
 [ 102 7274]]
Outcome values :
 140 122 102 7274
Classification report :
               precision    recall  f1-score   support

           1       0.58      0.53      0.56       262
           0       0.98      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.78      0.76      0.77      7638
weighted avg       0.97      0.97      0.97      7638

Accuracy : 97.1 %
Precision : 57.9 %
Recall : 53.4 %
F1 Score : 0.556
Specificity or True Negative Rate : 98.6 %
Balanced Accuracy : 76.0 %
MCC : 0.541
roc_auc_score: 0.76
```



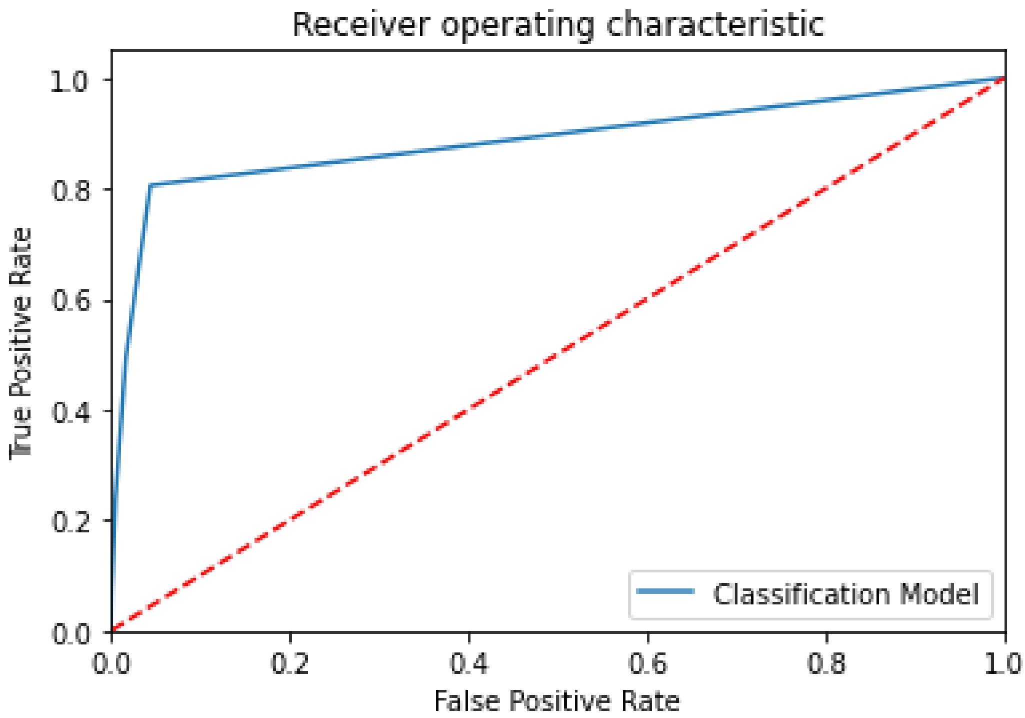

```
KNN_K_value =  2
Model Name:  KNeighborsClassifier(n_neighbors=2)
Confusion matrix :
 [[   63   199]
 [   43 7333]]
Outcome values :
 63 199 43 7333
Classification report :
               precision    recall  f1-score   support

           1       0.59      0.24      0.34       262
           0       0.97      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.78      0.62      0.66      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.8 %
Precision : 59.4 %
Recall : 24.0 %
F1 Score : 0.342
Specificity or True Negative Rate : 99.4 %
Balanced Accuracy : 61.7 %
MCC : 0.365
roc_auc_score: 0.617
```



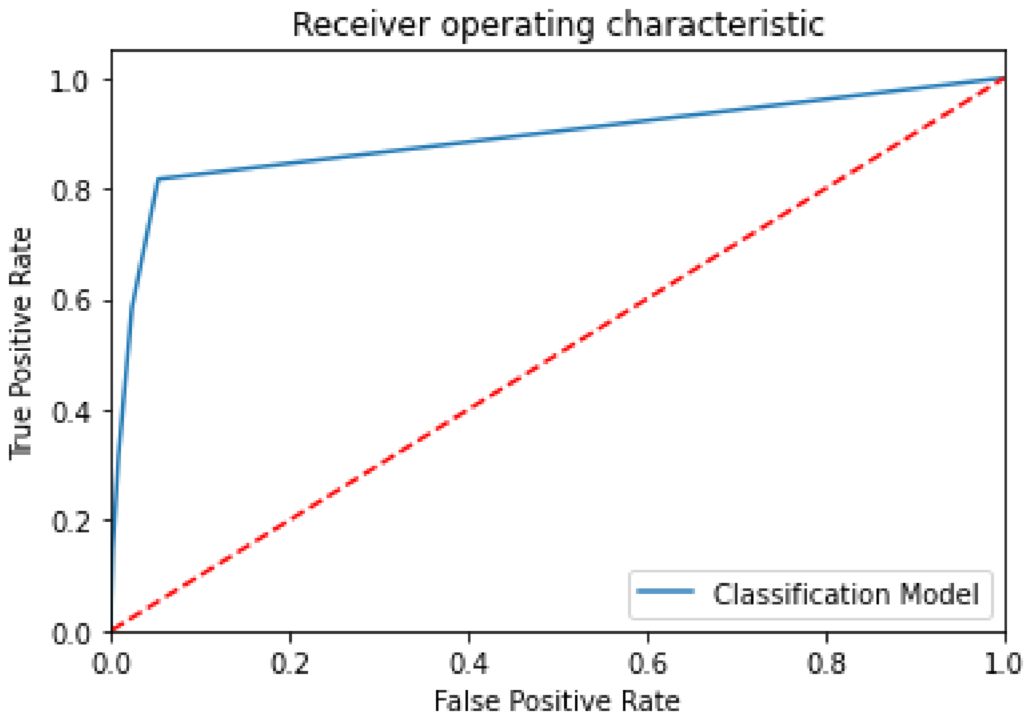

```
KNN_K_value =  3
Model Name:  KNeighborsClassifier(n_neighbors=3)
Confusion matrix :
 [[ 106  156]
 [  77 7299]]
Outcome values :
 106 156 77 7299
Classification report :
               precision    recall  f1-score   support

           1       0.58      0.40      0.48       262
           0       0.98      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.78      0.70      0.73      7638
weighted avg       0.97      0.97      0.97      7638

Accuracy : 96.9 %
Precision : 57.9 %
Recall : 40.5 %
F1 Score : 0.476
Specificity or True Negative Rate : 99.0 %
Balanced Accuracy : 69.8 %
MCC : 0.469
roc_auc_score: 0.697
```



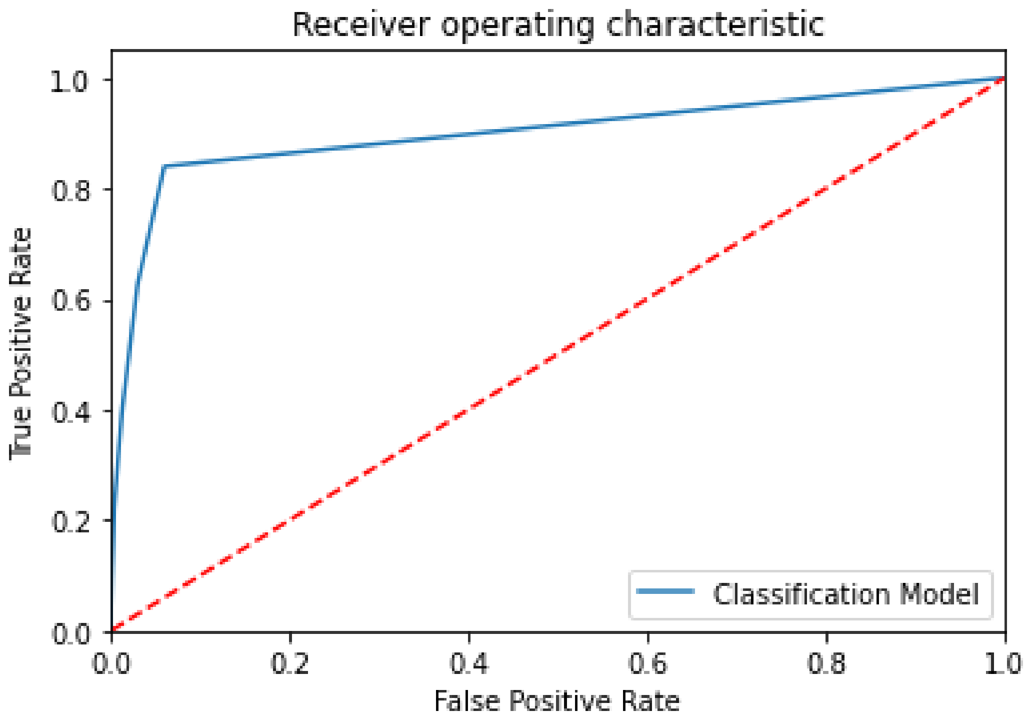

```
KNN_K_value =  4
Model Name:  KNeighborsClassifier(n_neighbors=4)
Confusion matrix :
 [[  62  200]
 [  35 7341]]
Outcome values :
 62 200 35 7341
Classification report :
               precision    recall  f1-score   support

           1       0.64      0.24      0.35       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.81      0.62      0.66      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 63.9 %
Recall : 23.7 %
F1 Score : 0.345
Specificity or True Negative Rate : 99.5 %
Balanced Accuracy : 61.6 %
MCC : 0.377
roc_auc_score: 0.616
```



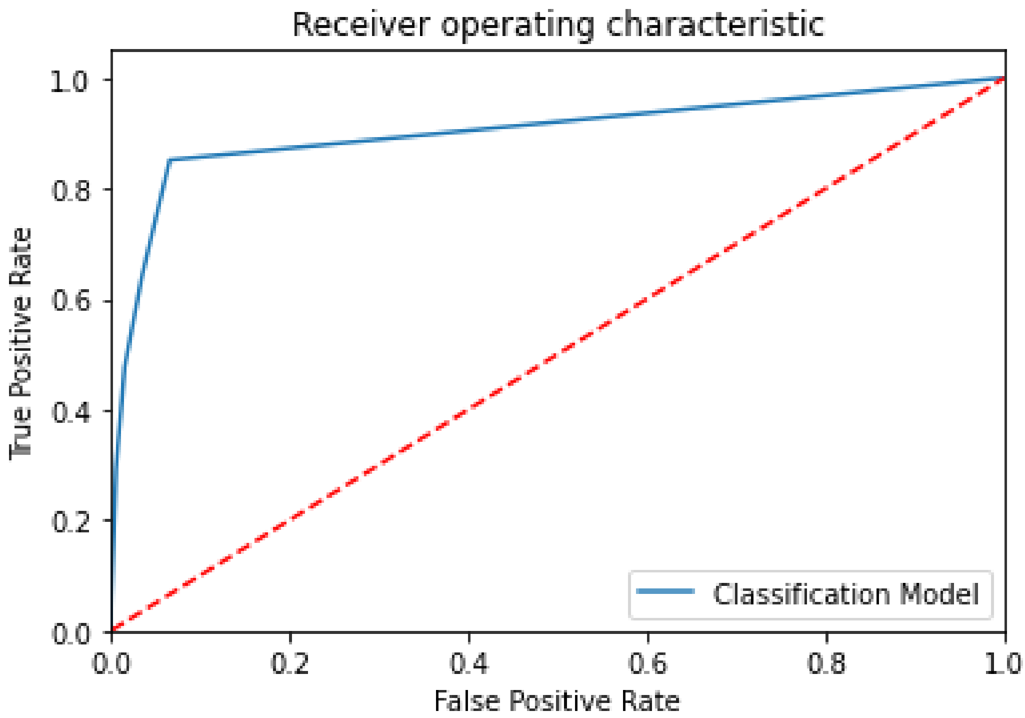

```
KNN_K_value =  5
Model Name:  KNeighborsClassifier()
Confusion matrix :
 [[  83  179]
 [  62 7314]]
Outcome values :
 83 179 62 7314
Classification report :
               precision    recall  f1-score   support

           1       0.57      0.32      0.41       262
           0       0.98      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.77      0.65      0.70      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.8 %
Precision : 57.2 %
Recall : 31.7 %
F1 Score : 0.408
Specificity or True Negative Rate : 99.2 %
Balanced Accuracy : 65.4 %
MCC : 0.411
roc_auc_score: 0.654
```



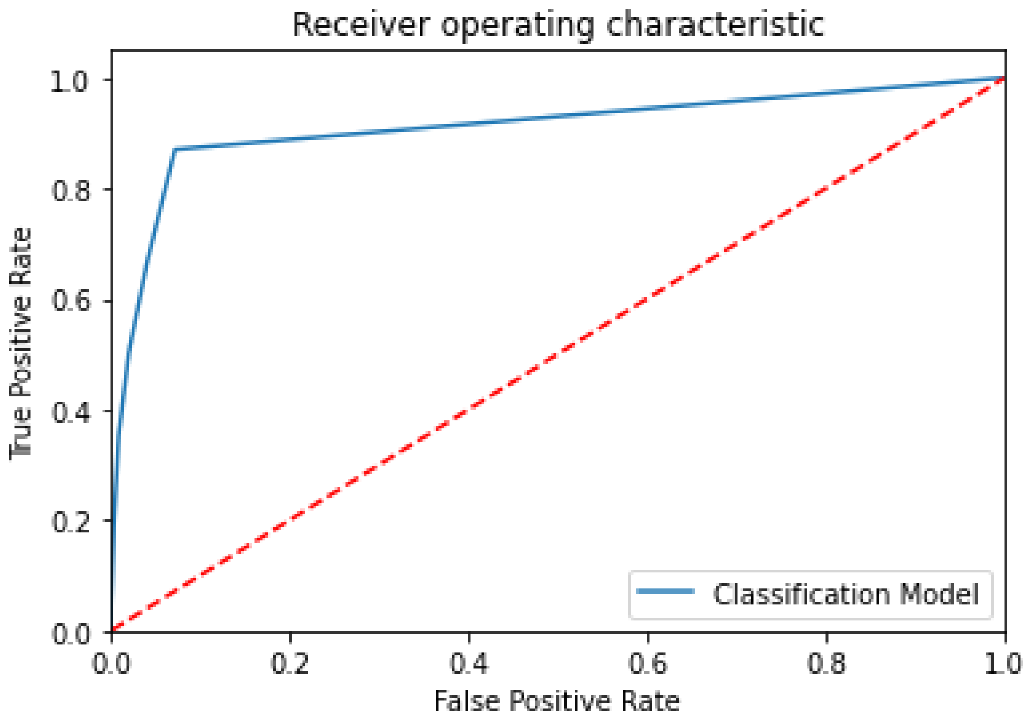

```
KNN_K_value =  6
Model Name:  KNeighborsClassifier(n_neighbors=6)
Confusion matrix :
 [[  58  204]
 [  25 7351]]
Outcome values :
 58 204 25 7351
Classification report :
               precision    recall  f1-score   support

           1       0.70      0.22      0.34       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.84      0.61      0.66      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 69.9 %
Recall : 22.1 %
F1 Score : 0.336
Specificity or True Negative Rate : 99.7 %
Balanced Accuracy : 60.9 %
MCC : 0.383
roc_auc_score: 0.609
```



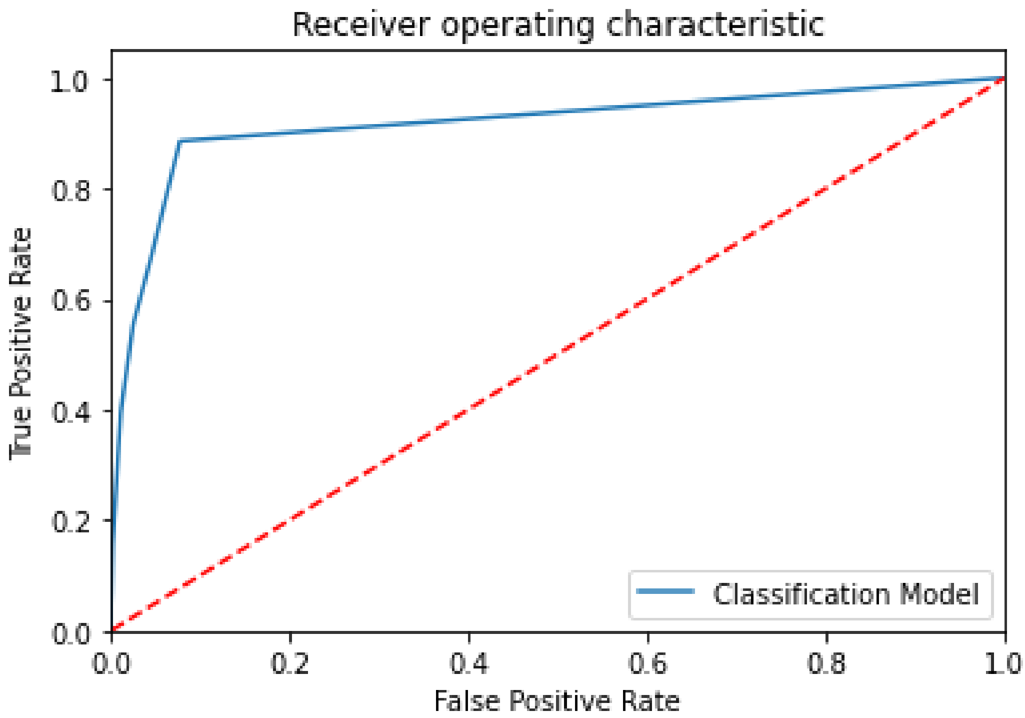

```
KNN_K_value =  7
Model Name:  KNeighborsClassifier(n_neighbors=7)
Confusion matrix :
 [[   77  185]
 [   41 7335]]
Outcome values :
 77 185 41 7335
Classification report :
               precision    recall  f1-score   support

           1       0.65      0.29      0.41       262
           0       0.98      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.81      0.64      0.70      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 65.3 %
Recall : 29.4 %
F1 Score : 0.405
Specificity or True Negative Rate : 99.4 %
Balanced Accuracy : 64.4 %
MCC : 0.426
roc_auc_score: 0.644
```



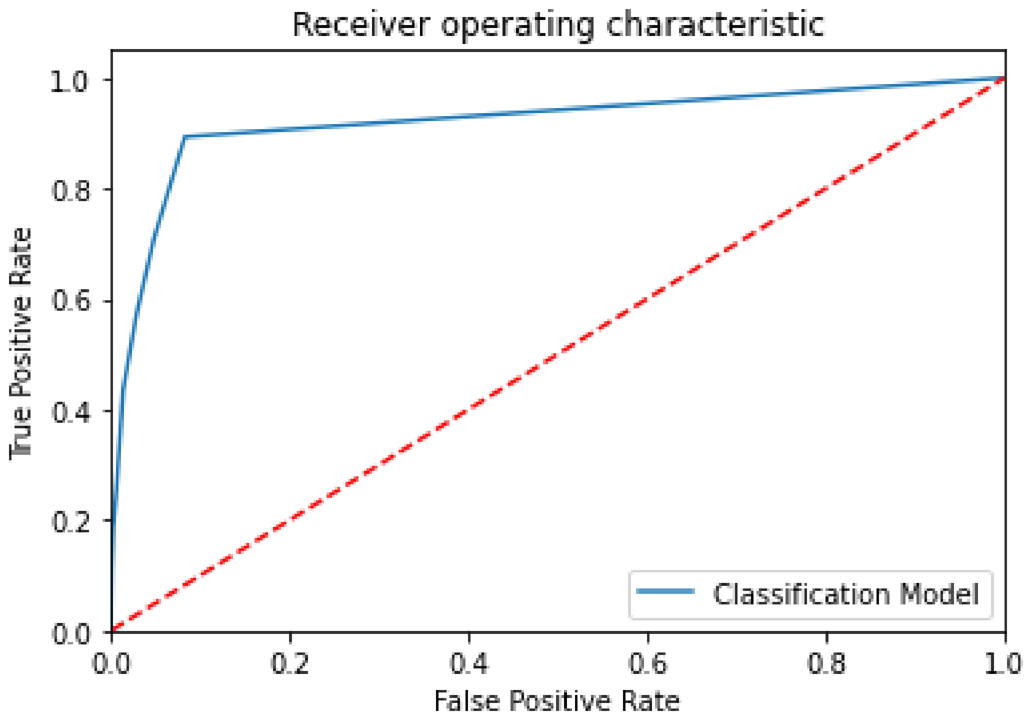

```
KNN_K_value =  8
Model Name:  KNeighborsClassifier(n_neighbors=8)
Confusion matrix :
 [[  50  212]
 [  22 7354]]
Outcome values :
 50 212 22 7354
Classification report :
               precision    recall  f1-score   support

           1       0.69      0.19      0.30       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.83      0.59      0.64      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 69.4 %
Recall : 19.1 %
F1 Score : 0.299
Specificity or True Negative Rate : 99.7 %
Balanced Accuracy : 59.4 %
MCC : 0.354
roc_auc_score: 0.594
```



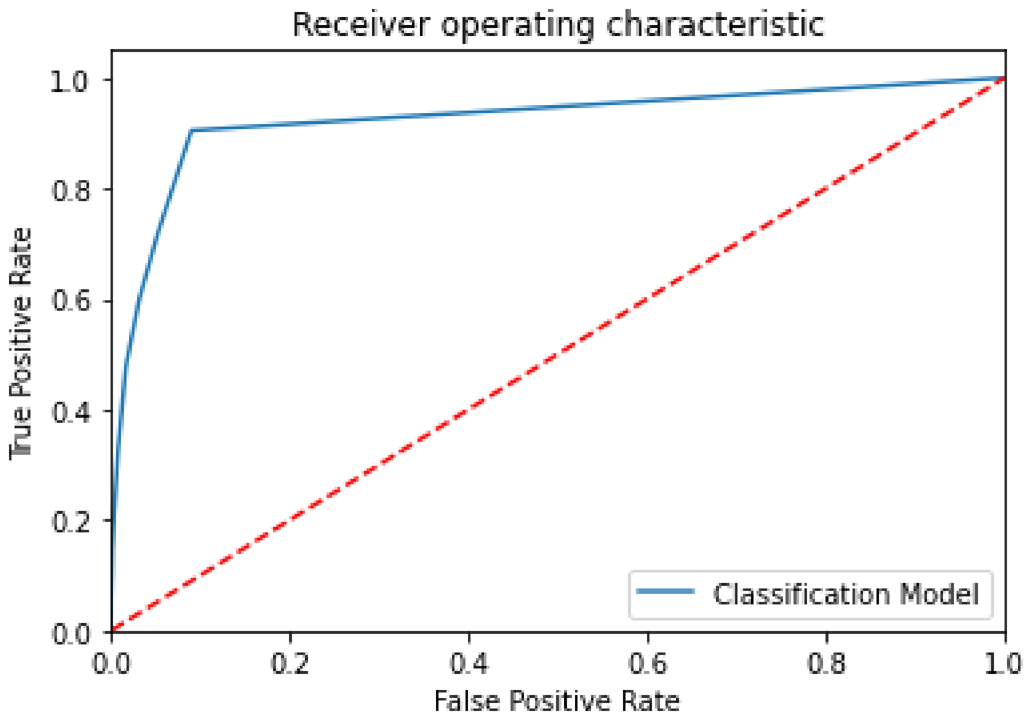

```
KNN_K_value =  9
Model Name:  KNeighborsClassifier(n_neighbors=9)
Confusion matrix :
 [[  58  204]
 [  31 7345]]
Outcome values :
 58 204 31 7345
Classification report :
               precision    recall  f1-score   support

           1       0.65      0.22      0.33       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.81      0.61      0.66      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 65.2 %
Recall : 22.1 %
F1 Score : 0.33
Specificity or True Negative Rate : 99.6 %
Balanced Accuracy : 60.8 %
MCC : 0.368
roc_auc_score: 0.609
```



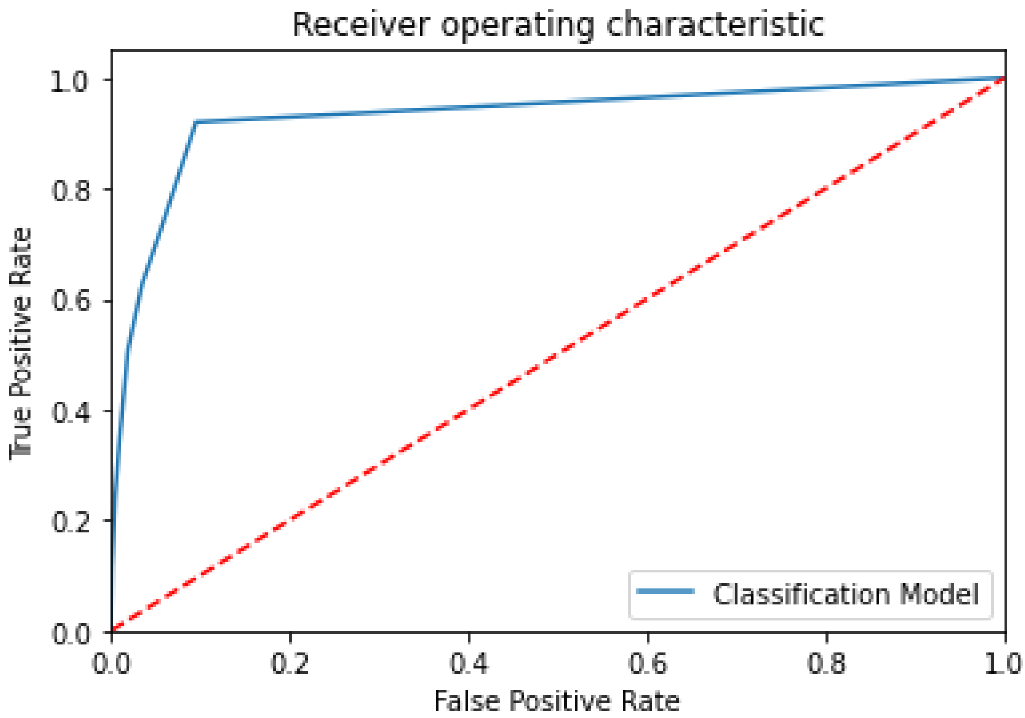

```
KNN_K_value =  10
Model Name:  KNeighborsClassifier(n_neighbors=10)
Confusion matrix :
 [[   48  214]
 [   16 7360]]
Outcome values :
 48 214 16 7360
Classification report :
               precision    recall  f1-score   support

           1       0.75      0.18      0.29       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.86      0.59      0.64      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 75.0 %
Recall : 18.3 %
F1 Score : 0.294
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 59.0 %
MCC : 0.361
roc_auc_score: 0.591
```



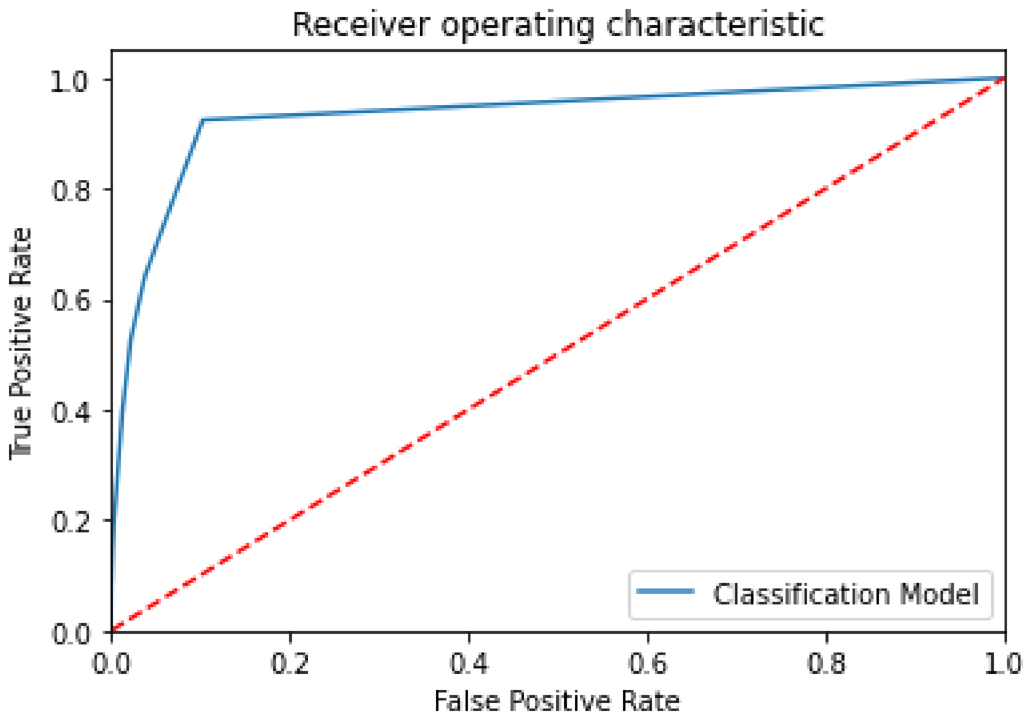

```
KNN_K_value =  11
Model Name:  KNeighborsClassifier(n_neighbors=11)
Confusion matrix :
 [[  55  207]
 [  24 7352]]
Outcome values :
 55 207 24 7352
Classification report :
               precision    recall  f1-score   support

           1       0.70      0.21      0.32       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.83      0.60      0.65      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 69.6 %
Recall : 21.0 %
F1 Score : 0.323
Specificity or True Negative Rate : 99.7 %
Balanced Accuracy : 60.4 %
MCC : 0.372
roc_auc_score: 0.603
```



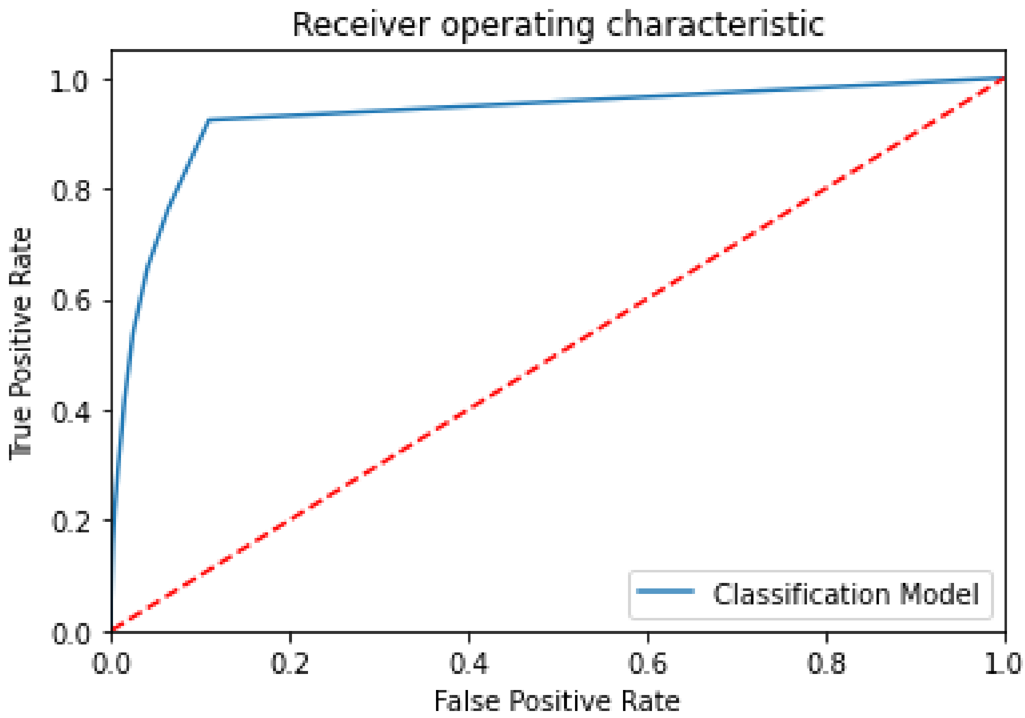

```
KNN_K_value =  12
Model Name:  KNeighborsClassifier(n_neighbors=12)
Confusion matrix :
 [[   45  217]
 [   18 7358]]
Outcome values :
 45 217 18 7358
Classification report :
               precision    recall  f1-score   support

           1       0.71      0.17      0.28       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.84      0.58      0.63      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 71.4 %
Recall : 17.2 %
F1 Score : 0.277
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 58.5 %
MCC : 0.341
roc_auc_score: 0.585
```



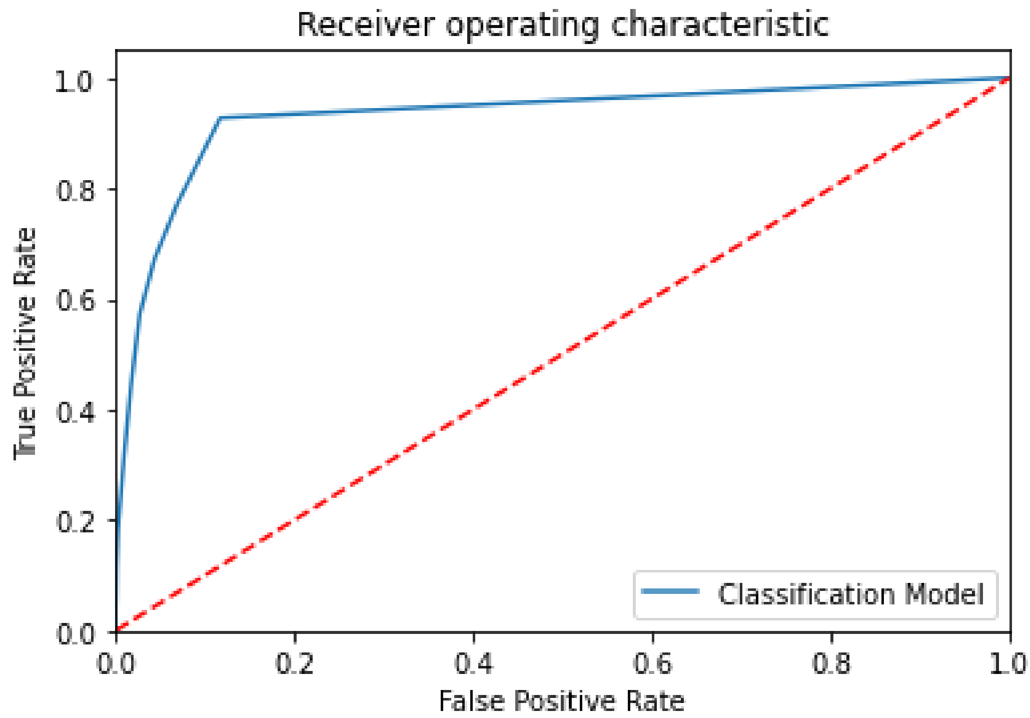

```
KNN_K_value =  13
Model Name:  KNeighborsClassifier(n_neighbors=13)
Confusion matrix :
 [[  50  212]
 [  21 7355]]
Outcome values :
 50 212 21 7355
Classification report :
               precision    recall  f1-score   support

           1       0.70      0.19      0.30       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.84      0.59      0.64      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 70.4 %
Recall : 19.1 %
F1 Score : 0.3
Specificity or True Negative Rate : 99.7 %
Balanced Accuracy : 59.4 %
MCC : 0.357
roc_auc_score: 0.594
```



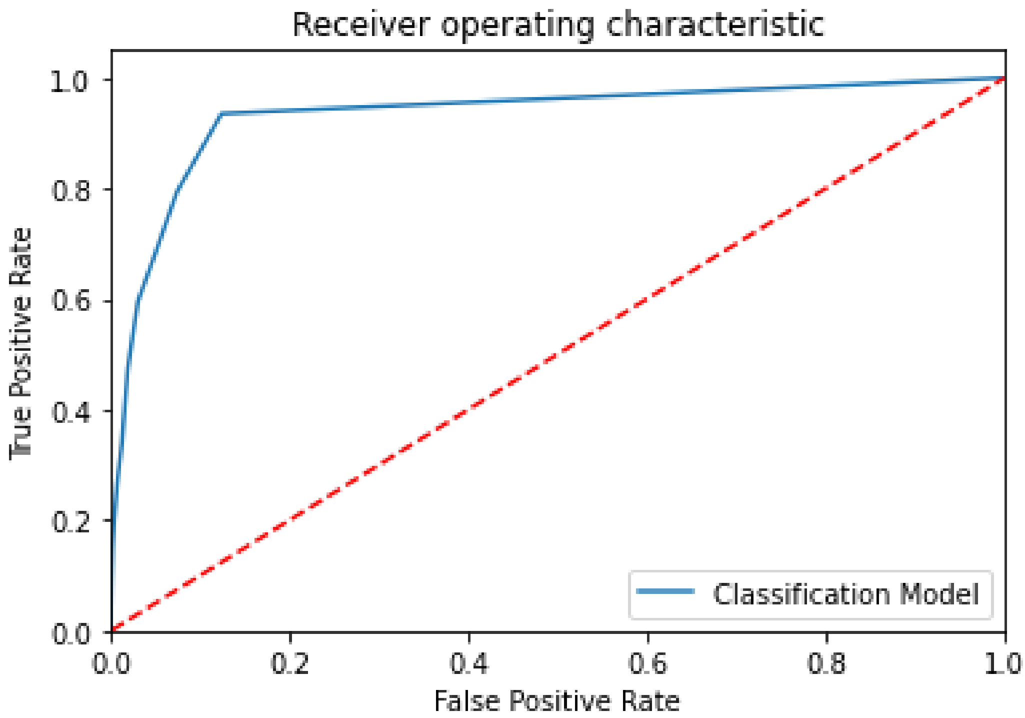

```
KNN_K_value =  14
Model Name:  KNeighborsClassifier(n_neighbors=14)
Confusion matrix :
 [[   41  221]
 [   13 7363]]
Outcome values :
 41 221 13 7363
Classification report :
               precision    recall  f1-score   support

           1       0.76      0.16      0.26       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.87      0.58      0.62      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 75.9 %
Recall : 15.6 %
F1 Score : 0.259
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 57.7 %
MCC : 0.336
roc_auc_score: 0.577
```



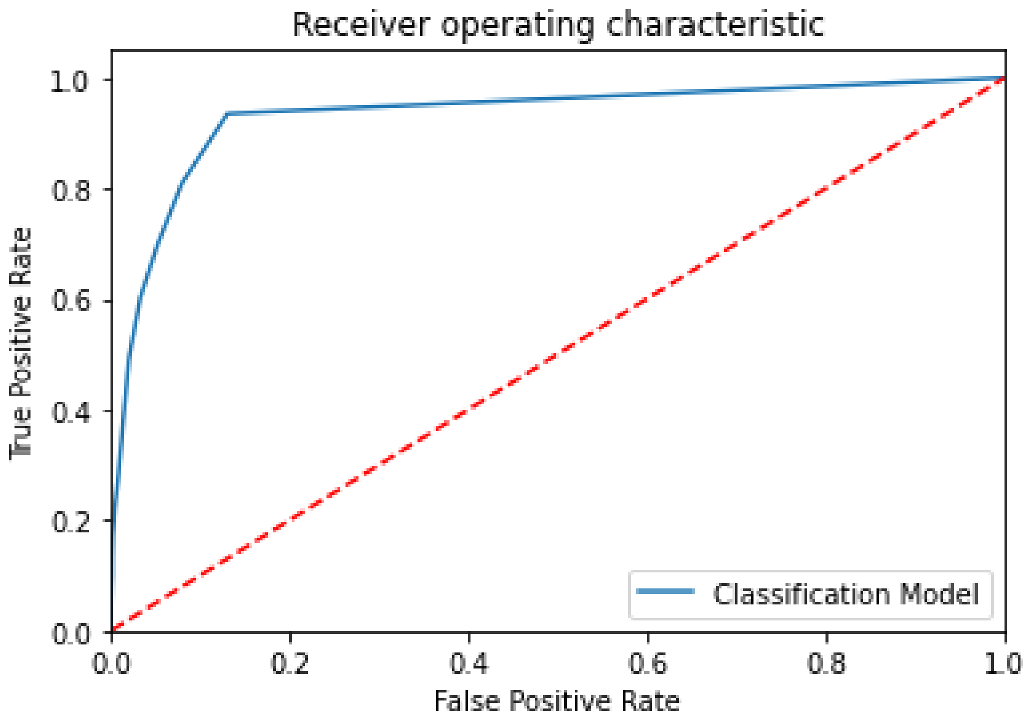

```
KNN_K_value =  15
Model Name:  KNeighborsClassifier(n_neighbors=15)
Confusion matrix :
 [[  48  214]
 [  18 7358]]
Outcome values :
 48 214 18 7358
Classification report :
               precision    recall  f1-score   support

           1       0.73      0.18      0.29       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.85      0.59      0.64      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 72.7 %
Recall : 18.3 %
F1 Score : 0.293
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 59.0 %
MCC : 0.355
roc_auc_score: 0.59
```



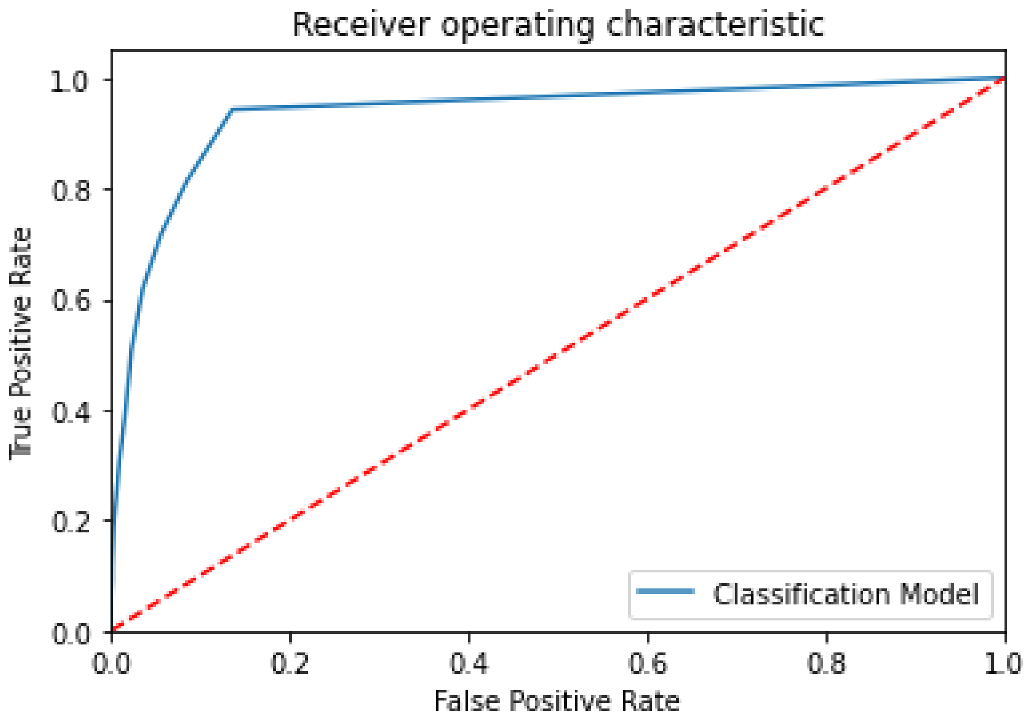

```
KNN_K_value =  16
Model Name:  KNeighborsClassifier(n_neighbors=16)
Confusion matrix :
 [[  40  222]
 [  12 7364]]
Outcome values :
 40 222 12 7364
Classification report :
               precision    recall  f1-score   support

           1       0.77      0.15      0.25       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.87      0.58      0.62      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 76.9 %
Recall : 15.3 %
F1 Score : 0.255
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 57.6 %
MCC : 0.334
roc_auc_score: 0.576
```



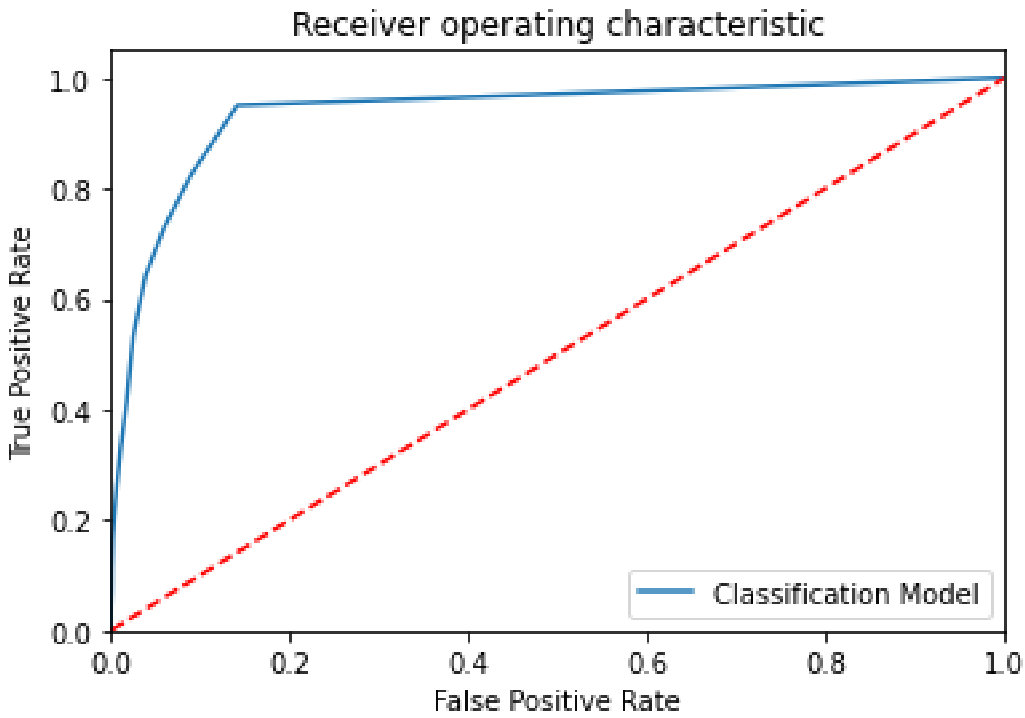

```
KNN_K_value =  17
Model Name:  KNeighborsClassifier(n_neighbors=17)
Confusion matrix :
 [[  45  217]
 [  13 7363]]
Outcome values :
 45 217 13 7363
Classification report :
               precision    recall  f1-score   support

           1       0.78      0.17      0.28       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.87      0.58      0.63      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 77.6 %
Recall : 17.2 %
F1 Score : 0.281
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 58.5 %
MCC : 0.356
roc_auc_score: 0.585
```



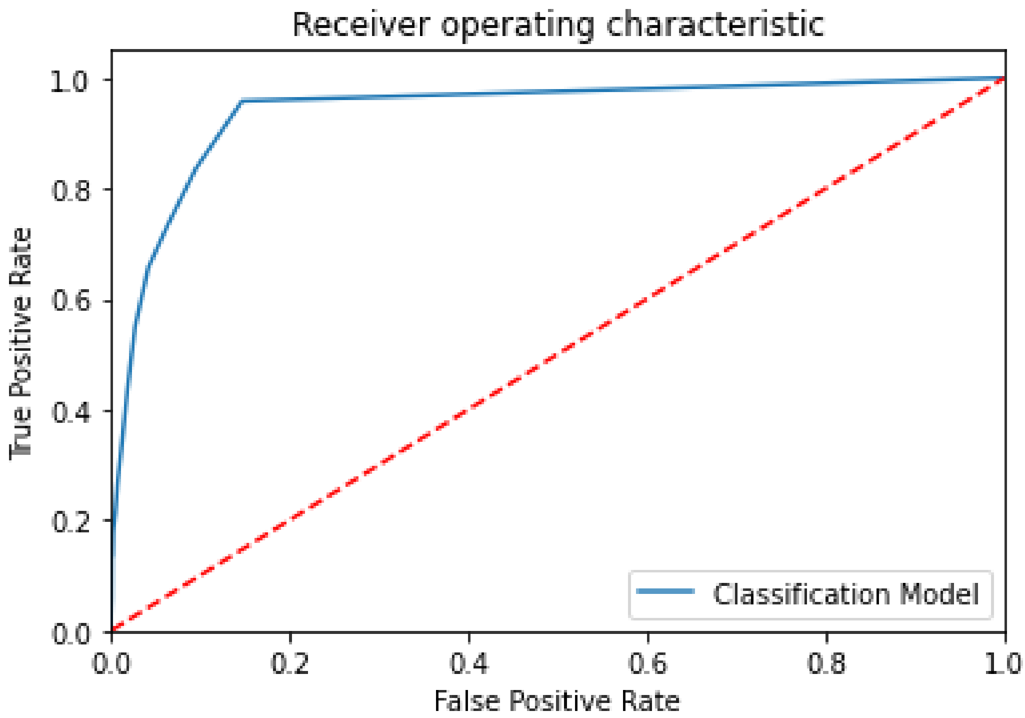

```
KNN_K_value =  18
Model Name:  KNeighborsClassifier(n_neighbors=18)
Confusion matrix :
 [[  38  224]
 [  12 7364]]
Outcome values :
 38 224 12 7364
Classification report :
               precision    recall  f1-score   support

           1       0.76      0.15      0.24       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.87      0.57      0.61      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 76.0 %
Recall : 14.5 %
F1 Score : 0.244
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 57.2 %
MCC : 0.324
roc_auc_score: 0.572
```




```
KNN_K_value =  19
Model Name:  KNeighborsClassifier(n_neighbors=19)
Confusion matrix :
 [[  44  218]
 [  14 7362]]
Outcome values :
 44 218 14 7362
Classification report :
               precision    recall  f1-score   support

           1       0.76      0.17      0.28       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.86      0.58      0.63      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 97.0 %
Precision : 75.9 %
Recall : 16.8 %
F1 Score : 0.275
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 58.3 %
MCC : 0.348
roc_auc_score: 0.583
```




```
KNN_K_value =  20
Model Name:  KNeighborsClassifier(n_neighbors=20)
Confusion matrix :
 [[  36  226]
 [  12 7364]]
Outcome values :
 36 226 12 7364
Classification report :
               precision    recall  f1-score   support

           1       0.75      0.14      0.23       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.86      0.57      0.61      7638
weighted avg       0.96      0.97      0.96      7638

Accuracy : 96.9 %
Precision : 75.0 %
Recall : 13.7 %
F1 Score : 0.232
Specificity or True Negative Rate : 99.8 %
Balanced Accuracy : 56.8 %
MCC : 0.313
roc_auc_score: 0.568
```



In [59]:

```
# display the KNNResults
KNNResults
```

Out[59]:

	Model Name	KNNK Value	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	KNeighborsClassifier (n_neighbors=1)	1	140	122	102	7274	0.971	0.579	0.534	0.556	0.986	0.541	0.760261	0.76
1	KNeighborsClassifier (n_neighbors=2)	2	63	199	43	7333	0.968	0.594	0.24	0.342	0.994	0.365	0.617314	0.617
2	KNeighborsClassifier (n_neighbors=3)	3	106	156	77	7299	0.969	0.579	0.405	0.476	0.99	0.469	0.69707	0.698
3	KNeighborsClassifier (n_neighbors=4)	4	62	200	35	7341	0.969	0.639	0.237	0.345	0.995	0.377	0.615948	0.616
4	KNeighborsClassifier ()	5	83	179	62	7314	0.968	0.572	0.317	0.408	0.992	0.411	0.654194	0.654
5	KNeighborsClassifier (n_neighbors=6)	6	58	204	25	7351	0.97	0.699	0.221	0.336	0.997	0.383	0.608992	0.609
6	KNeighborsClassifier (n_neighbors=7)	7	77	185	41	7335	0.97	0.653	0.294	0.405	0.994	0.426	0.644167	0.644

	Model Name	KNNK Value	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
7	KNeighborsClassifier (n_neighbors=8)	8	50	212	22	7354	0.969	0.694	0.191	0.299	0.997	0.354	0.593929	0.594
8	KNeighborsClassifier (n_neighbors=9)	9	58	204	31	7345	0.969	0.652	0.221	0.33	0.996	0.368	0.608586	0.608
9	KNeighborsClassifier (n_neighbors=10)	10	48	214	16	7360	0.97	0.75	0.183	0.294	0.998	0.361	0.590518	0.59
10	KNeighborsClassifier (n_neighbors=11)	11	55	207	24	7352	0.97	0.696	0.21	0.323	0.997	0.372	0.603335	0.604
11	KNeighborsClassifier (n_neighbors=12)	12	45	217	18	7358	0.969	0.714	0.172	0.277	0.998	0.341	0.584658	0.585
12	KNeighborsClassifier (n_neighbors=13)	13	50	212	21	7355	0.969	0.704	0.191	0.3	0.997	0.357	0.593996	0.594
13	KNeighborsClassifier (n_neighbors=14)	14	41	221	13	7363	0.969	0.759	0.156	0.259	0.998	0.336	0.577363	0.577
14	KNeighborsClassifier (n_neighbors=15)	15	48	214	18	7358	0.97	0.727	0.183	0.293	0.998	0.355	0.590383	0.59
15	KNeighborsClassifier (n_neighbors=16)	16	40	222	12	7364	0.969	0.769	0.153	0.255	0.998	0.334	0.575522	0.576
16	KNeighborsClassifier (n_neighbors=17)	17	45	217	13	7363	0.97	0.776	0.172	0.281	0.998	0.356	0.584997	0.585
17	KNeighborsClassifier (n_neighbors=18)	18	38	224	12	7364	0.969	0.76	0.145	0.244	0.998	0.324	0.571706	0.572
18	KNeighborsClassifier (n_neighbors=19)	19	44	218	14	7362	0.97	0.759	0.168	0.275	0.998	0.348	0.58302	0.583
19	KNeighborsClassifier (n_neighbors=20)	20	36	226	12	7364	0.969	0.75	0.137	0.232	0.998	0.313	0.567889	0.568

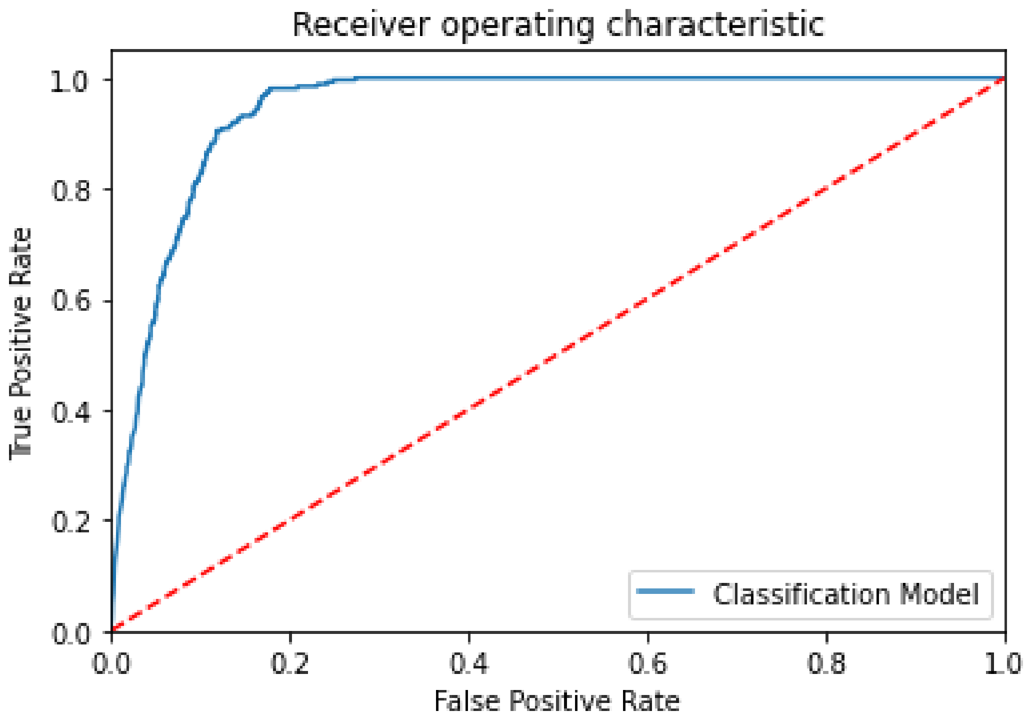
Naive Bayes Model(Guassianb)Algorithm

In [60]:

```
# Training the Naive Bayes model (GaussianNB) on the Training set
```

```python
from sklearn.naive_bayes import GaussianNB

modelGNB = GaussianNB(priors=None, var_smoothing=1e-09)

# Fit the model with train data

modelGNB.fit(x_train,y_train)

# Predict the model with test data set

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

# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual,modelGNB.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' % model_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('-----------------------------------------------------------------------------------------------
--------------')
Confusion matrix :
 [[ 262    0]
 [2131 5245]]
Outcome values :
 262 0 2131 5245
Classification report :
              precision    recall  f1-score   support

           1       0.11      1.00      0.20       262
           0       1.00      0.71      0.83      7376

    accuracy                           0.72      7638
   macro avg       0.55      0.86      0.51      7638
weighted avg       0.97      0.72      0.81      7638

Accuracy : 72.1 %
Precision : 10.9 %
Recall : 100.0 %
F1 Score : 0.197
Specificity or True Negative Rate : 71.1 %
Balanced Accuracy : 85.5 %
MCC : 0.279
roc_auc_score: 0.856
```



Support Vector Machines_Linear Kernal (SVM)

In [61]:

```
# Load the results file for  SVM

EMResults1 = pd.read_excel(r"EMResultsNew.xlsx", header=0)

# Display the first 5 records

EMResults1.head()
```

Out[61]:

Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy

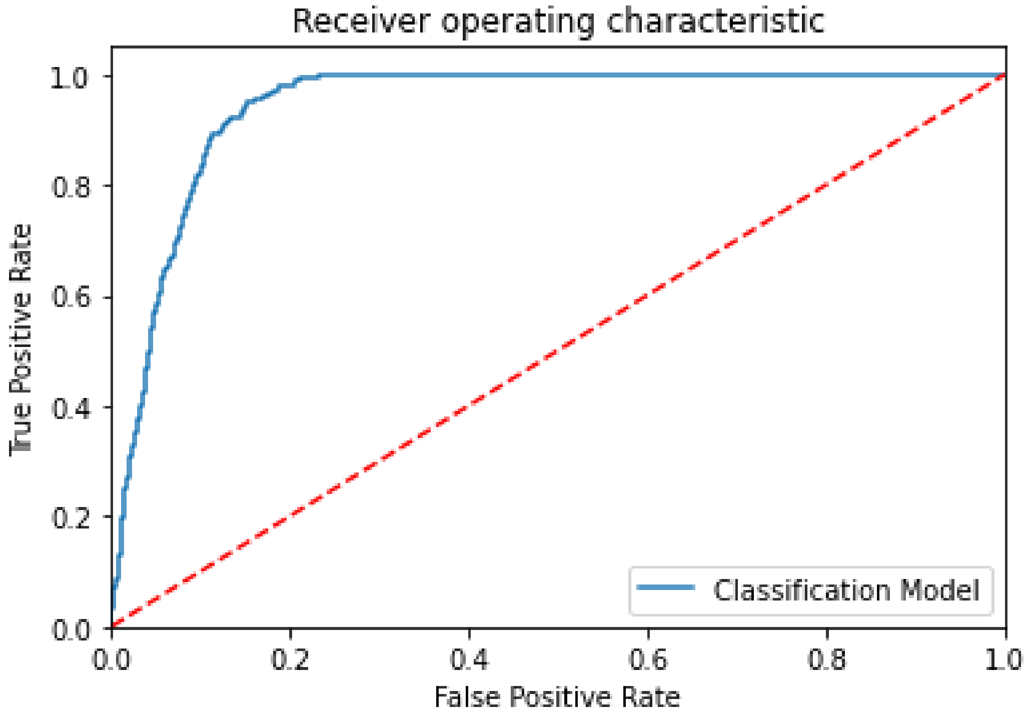
In [62]:

```
# Training the SVM algorithm with train dataset

from sklearn.svm import SVC

ModelSVM1 = SVC(C=1.0, kernel='linear', degree=3, gamma='scale', coef0=0.0, shrinking=True,
                probability=True, tol=0.001, cache_size=200, class_weight=None, verbose=False,
                max_iter=- 1, decision_function_shape='ovr', break_ties=False,
random_state=None)

# Train the model with train data

ModelSVM1 = ModelSVM1.fit(x_train, y_train)

# Predict the model with test data set

y_pred = ModelSVM1.predict(x_test)
y_pred_prob = ModelSVM1.predict_proba(x_test)
```

```python
# Print the model name

print('Model Name: ', "SVM - Linear")

# 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
model_roc_auc = roc_auc_score(actual, predicted)
fpr, tpr, thresholds = roc_curve(actual,ModelSVM1.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' % model_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' : "SVM - Linear",
            '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}
EMResults1 = EMResults1.append(new_row, ignore_index=True)
#-----------------------------------------------------------------------------------------------
----------------
Model Name:  SVM - Linear
Confusion matrix :
 [[   0  262]
 [   0 7376]]
Outcome values :
 0 262 0 7376
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.48      0.50      0.49      7638
weighted avg       0.93      0.97      0.95      7638

Accuracy : 96.6 %
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
```




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

In [63]:

```
# display the EMResults
EMResults1.head()
```

Out[63]:

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	SVM - Linear	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5

SVM_Polinomial Kernal

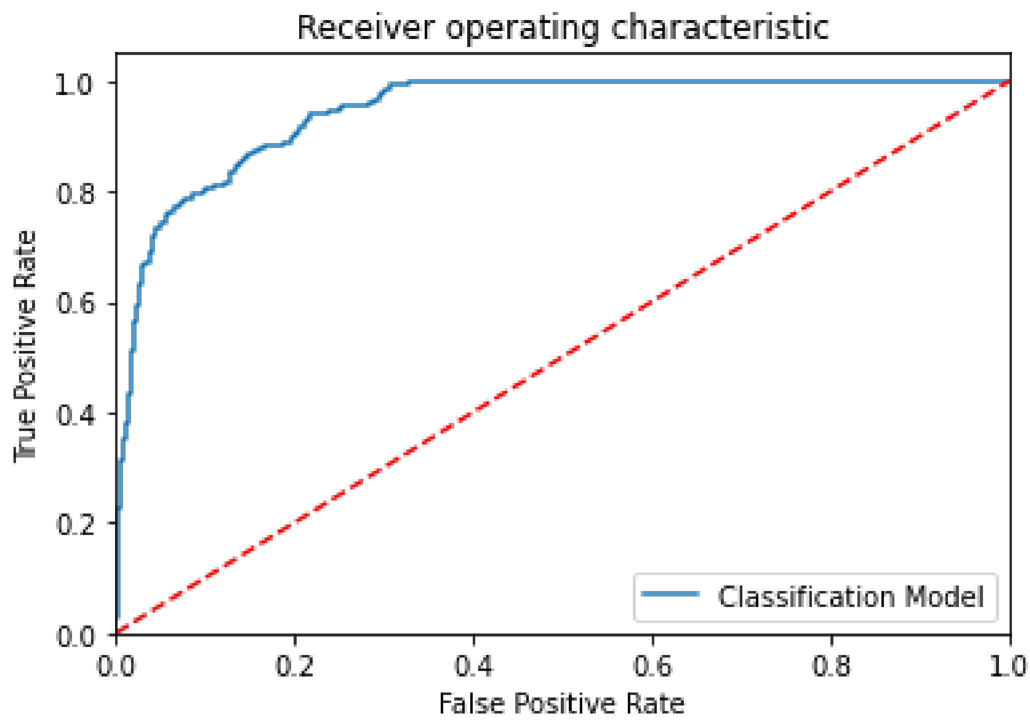
In [64]:

```
# Training the SVM algorithm

from sklearn.svm import SVC

ModelSVMPoly = SVC(kernel='poly', degree=2, probability=True)

# Train the model

ModelSVMPoly.fit(x_train, y_train)

# Predict the model with test data set

y_pred = ModelSVMPoly.predict(x_test)
y_pred_prob = ModelSVMPoly.predict_proba(x_test)
```

```
# Print the model name

print('Model Name: ', "SVM - Polynominal")

# 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(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelSVMPoly.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot
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' : "SVM - Polynominal",
            '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}
EMResults1 = EMResults1.append(new_row, ignore_index=True)
#-----------------------------------------------------------------------------------------------
--
Model Name:  SVM - Polynominal
Confusion matrix :
 [[   0  262]
 [   0 7376]]
Outcome values :
 0 262 0 7376
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.48      0.50      0.49      7638
weighted avg       0.93      0.97      0.95      7638

Accuracy : 96.6 %
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
```




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

In [65]:

```
# display the EMResults
EMResults1.head()
```

Out[65]:

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	SVM - Linear	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
1	SVM - Polynominal	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5

Gussain Kernal

In [66]:

```
# Training the SVM algorithm

from sklearn.svm import SVC

ModelSVMGaussian = SVC(kernel='rbf', random_state = 42, class_weight='balanced',
probability=True)

# Train the model

ModelSVMGaussian.fit(x_train, y_train)

# Predict the model with test data set
```

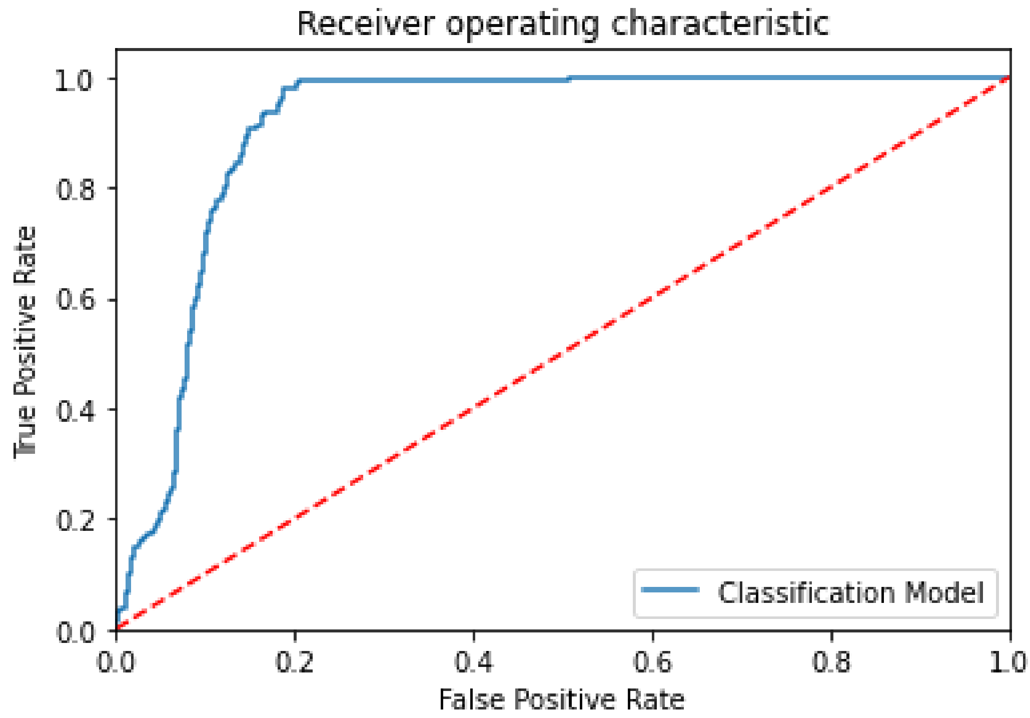
```python
y_pred = ModelSVMGaussian.predict(x_test)
y_pred_prob = ModelSVMGaussian.predict_proba(x_test)

# Confusion matrix in sklearn

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

# Print the model name

print('Model Name: ', "SVM - Gaussian")

# 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(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelSVMGaussian.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot
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' : "SVM - Gaussian",
            '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}
EMResults1 = EMResults1.append(new_row, ignore_index=True)
#----------------------------------------------------------------------------------------------------------
Model Name:  SVM - Gaussian
Confusion matrix :
 [[ 257    5]
 [1415 5961]]
Outcome values :
 257 5 1415 5961
Classification report :
```

	precision	recall	f1-score	support
1	0.15	0.98	0.27	262
0	1.00	0.81	0.89	7376
accuracy			0.81	7638
macro avg	0.58	0.89	0.58	7638
weighted avg	0.97	0.81	0.87	7638

```
Accuracy : 81.4 %
Precision : 15.4 %
Recall : 98.1 %
F1 Score : 0.266
```

```
Specificity or True Negative Rate : 80.8 %
Balanced Accuracy : 89.5 %
MCC : 0.347
roc_auc_score: 0.895
```



In [67]:

```
# display the EMResults
EMResults1.head()
```

Out[67]:

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	SVM - Linear	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
1	SVM - Polynominal	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
2	SVM - Gaussian	257	5	1415	5961	0.814	0.154	0.981	0.266	0.808	0.347	0.894539	0.895

Sigmoid Kernal

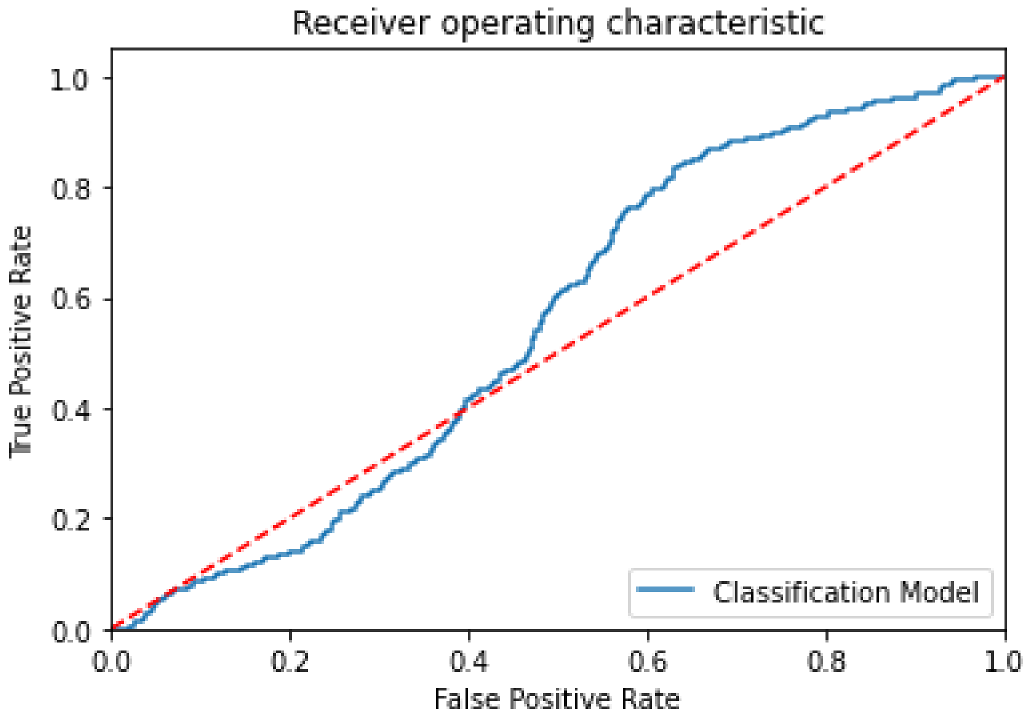
In [68]:

```
# Training the SVM algorithm

from sklearn.svm import SVC
```

```python
ModelSVMSig = SVC(kernel='sigmoid', random_state = 42, class_weight='balanced',
probability=True)

# Train the model

ModelSVMSig.fit(x_train, y_train)

# Predict the model with test data set

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

# Print the model name

print('Model Name: ', "SVM - Sigmoid")

# 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(y_test, y_pred), 3))

# ROC Curve

from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
fpr, tpr, thresholds = roc_curve(y_test,ModelSVMSig.predict_proba(x_test)[:,1])
plt.figure()
# plt.plot
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' : "SVM - Sigmoid",
            '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}
EMResults1 = EMResults1.append(new_row, ignore_index=True)
#-----------------------------------------------------------------------------------------------
--------------
Model Name:  SVM - Sigmoid
Confusion matrix :
 [[ 158  104]
 [3675 3701]]
Outcome values :
 158 104 3675 3701
Classification report :
               precision    recall  f1-score   support

           1       0.04      0.60      0.08       262
           0       0.97      0.50      0.66      7376
```

```
    accuracy                           0.51      7638
   macro avg       0.51      0.55      0.37      7638
weighted avg       0.94      0.51      0.64      7638

Accuracy : 50.5 %
Precision : 4.1 %
Recall : 60.3 %
F1 Score : 0.077
Specificity or True Negative Rate : 50.2 %
Balanced Accuracy : 55.2 %
MCC : 0.038
roc_auc_score: 0.552
```




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

In [69]:

```
# display the EMRseults
EMResults1.head()
```

Out[69]:

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	SVM - Linear	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
1	SVM - Polynomial	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
2	SVM - Gaussian	257	5	1415	5961	0.814	0.154	0.981	0.266	0.808	0.347	0.894539	0.895

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
3	SVM - Sigmoid	158	104	3675	3701	0.505	0.041	0.603	0.077	0.502	0.038	0.552408	0.552

Compare with the Classification algorithm

In [70]:

```
# load the concrete dataset

EMResults=pd.read_excel(r"EMResultsNew.xlsx",header=0)


# display the first 5 records
EMResults.head(10)
```

Out[70]:

Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy

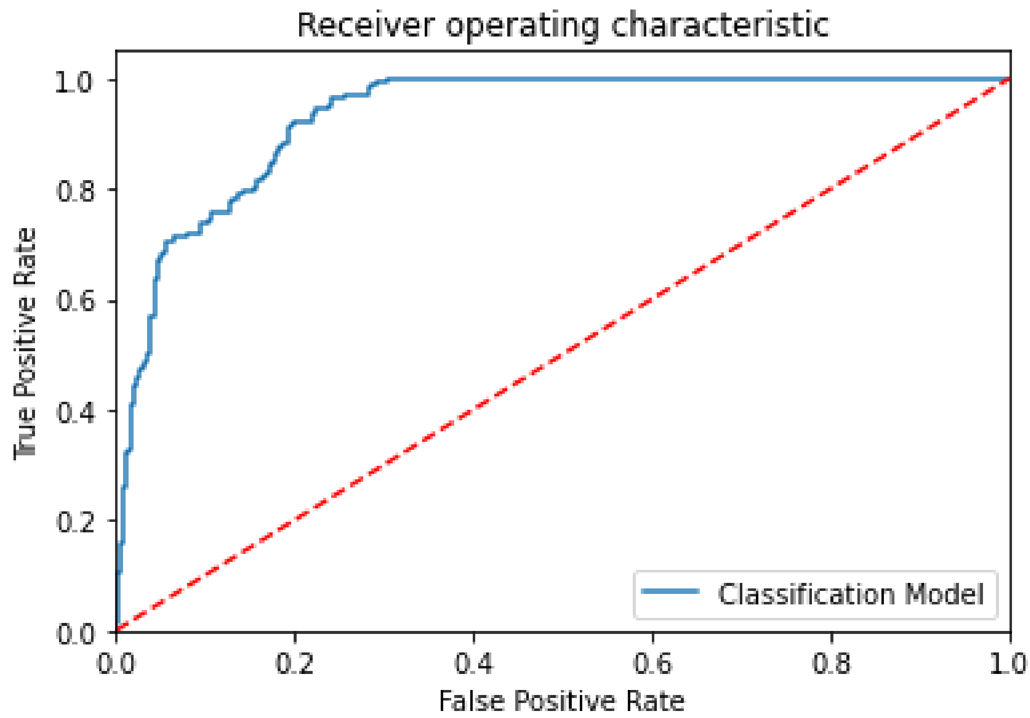
In [86]:

```
# 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

# Create objects of classification algorithms with default hyper-parameters

ModelLR = LogisticRegression()
ModelDC = DecisionTreeClassifier()
ModelRF = RandomForestClassifier()
ModelET = ExtraTreesClassifier()
ModelKNN = KNeighborsClassifier(n_neighbors=1)
ModelGNB = GaussianNB()
ModelSVMGaussian = SVC(kernel='rbf', random_state = 42, class_weight='balanced',
probability=True)

# Evalution matrix for all the algorithms

#MM = [ModelLR, ModelDC, ModelRF, ModelET, ModelKNN, ModelGNB, ModelSVM]
MM = [ModelLR, ModelDC, ModelRF, ModelET, ModelKNN, ModelGNB, ModelSVM]
for models in MM:

    # Train the model training dataset

    models.fit(x_train, y_train)

    # Prediction the model with test dataset
```

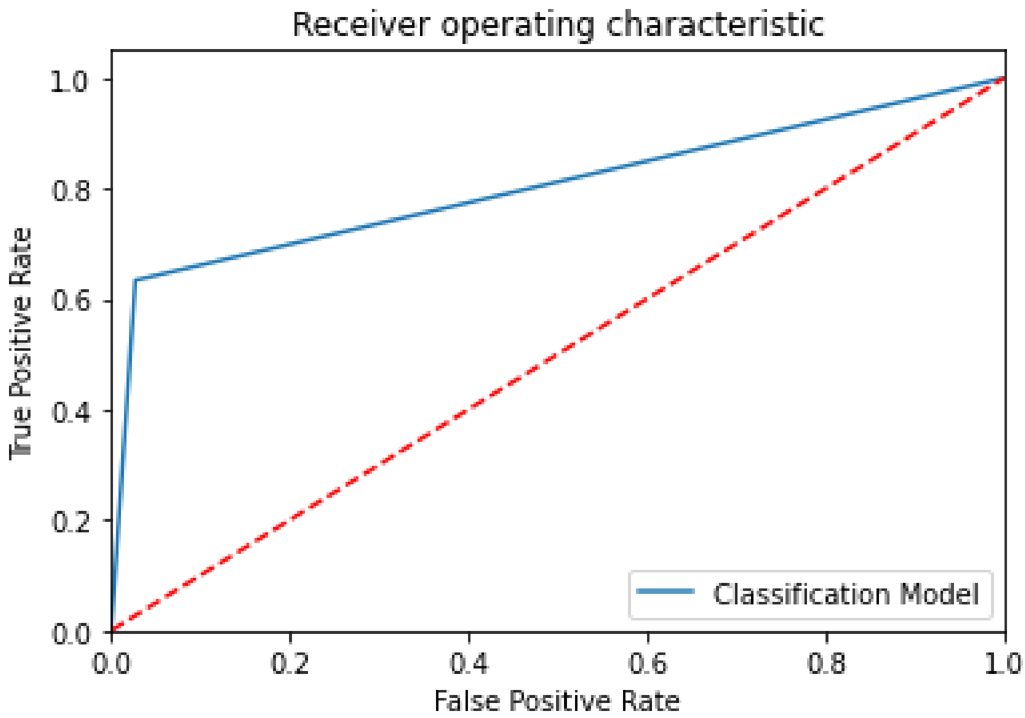
```
    y_pred = models.predict(x_test)
    y_pred_prob = models.predict_proba(x_test)

    # 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)

    # 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
    Model_roc_auc = roc_auc_score(actual, predicted)
    fpr, tpr, thresholds = roc_curve(actual, models.predict_proba(x_test)[:,1])
    plt.figure()
    #
    plt.plot(fpr, tpr, label= 'Classification Model' % Model_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}
    EMResults = EMResults.append(new_row, ignore_index=True)
    #-----------------------------------------------------------------------------------
Model Name:  LogisticRegression()
Confusion matrix :
 [[   0  262]
 [   0 7376]]
Outcome values :
 0 262 0 7376
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.48      0.50      0.49      7638
weighted avg       0.93      0.97      0.95      7638

Accuracy : 96.6 %
Precision : nan %
```

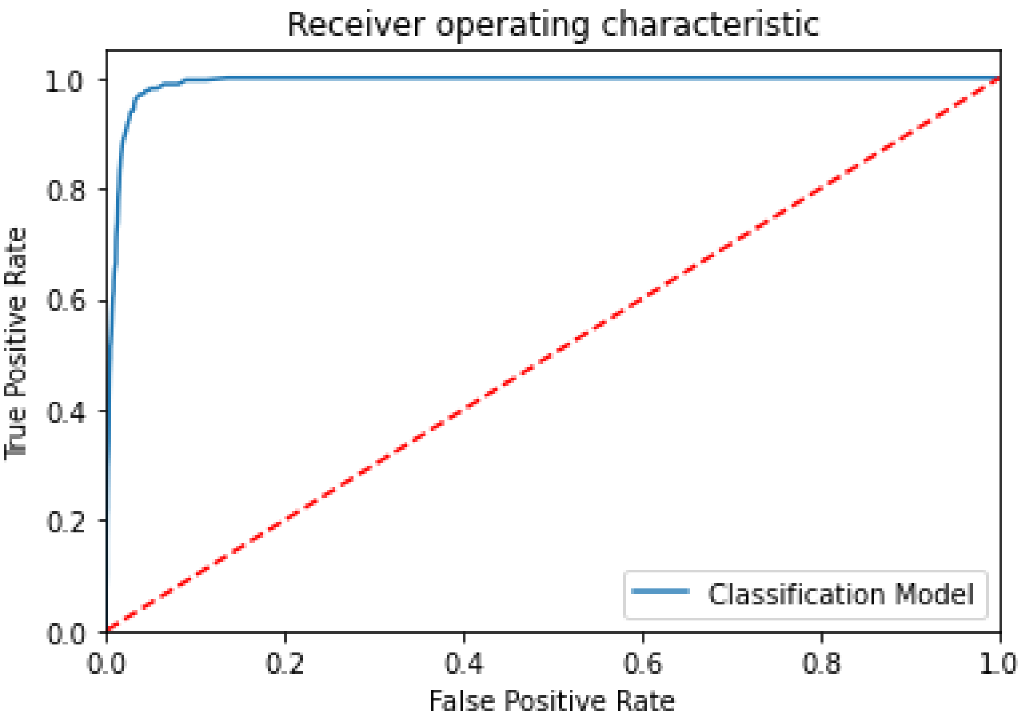
```
Recall : 0.0 %
F1 Score : 0.0
Specificity or True Negative Rate : 100.0 %
Balanced Accuracy : 50.0 %
MCC : nan
roc_auc_score: 0.5
```



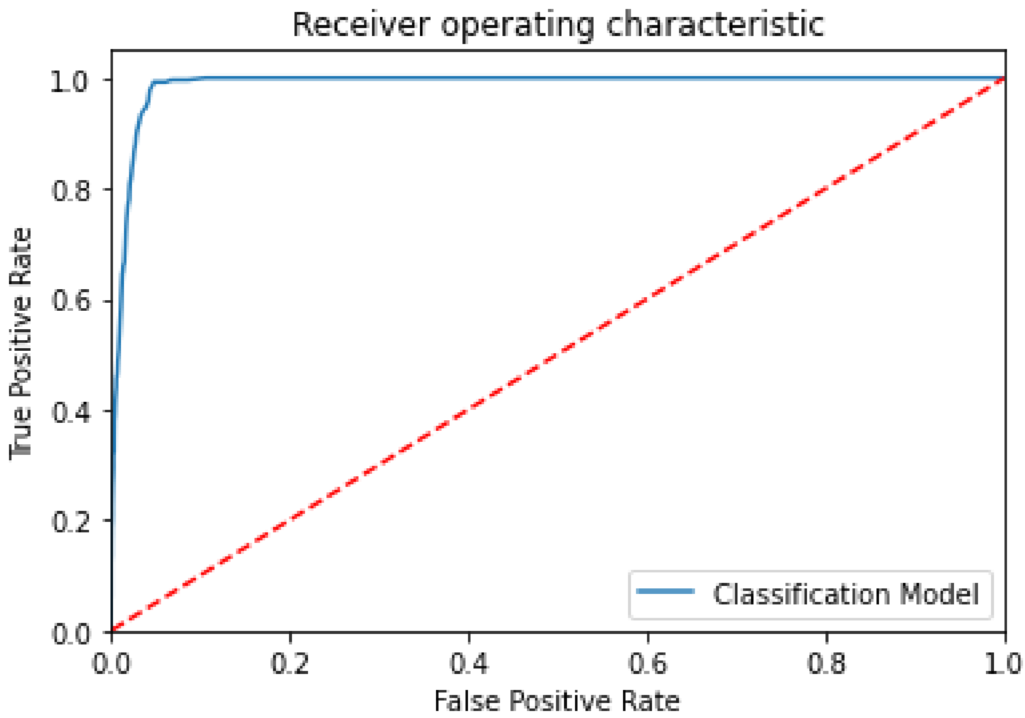

```
------------------------------------------------------------------------------------------------
-------
Model Name:  DecisionTreeClassifier()
Confusion matrix :
 [[ 166   96]
 [ 201 7175]]
Outcome values :
 166 96 201 7175
Classification report :
               precision    recall  f1-score   support

           1       0.45      0.63      0.53       262
           0       0.99      0.97      0.98      7376

    accuracy                           0.96      7638
   macro avg       0.72      0.80      0.75      7638
weighted avg       0.97      0.96      0.96      7638

Accuracy : 96.1 %
Precision : 45.2 %
Recall : 63.4 %
F1 Score : 0.528
Specificity or True Negative Rate : 97.3 %
Balanced Accuracy : 80.4 %
MCC : 0.516
roc_auc_score: 0.803
```



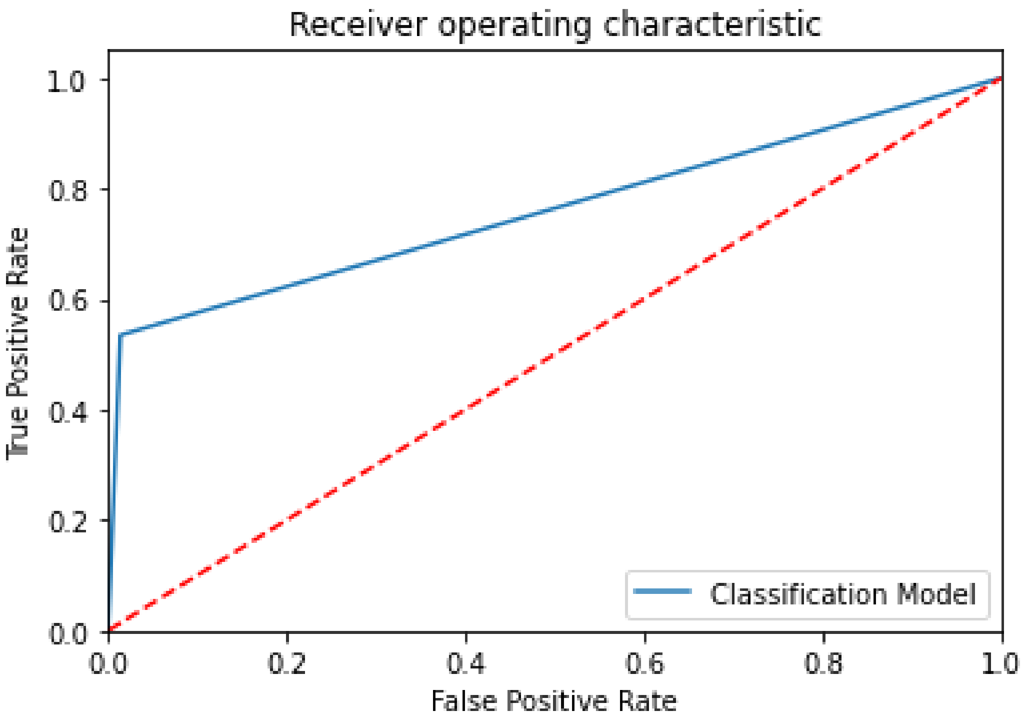

```
--------------------------------------------------------------------------------------------------
-------
Model Name:  RandomForestClassifier()
Confusion matrix :
 [[ 136  126]
 [  38 7338]]
Outcome values :
 136 126 38 7338
Classification report :
               precision    recall  f1-score   support

           1       0.78      0.52      0.62       262
           0       0.98      0.99      0.99      7376

    accuracy                           0.98      7638
   macro avg       0.88      0.76      0.81      7638
weighted avg       0.98      0.98      0.98      7638

Accuracy : 97.9 %
Precision : 78.2 %
Recall : 51.9 %
F1 Score : 0.624
Specificity or True Negative Rate : 99.5 %
Balanced Accuracy : 75.7 %
MCC : 0.627
roc_auc_score: 0.757
```



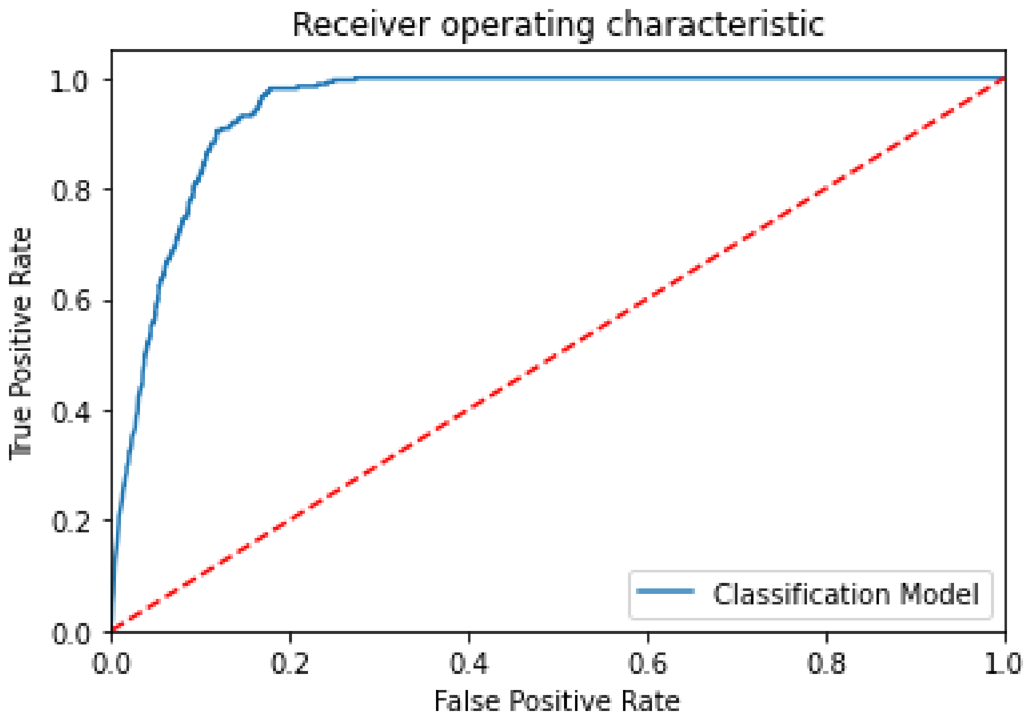

```
----------------------------------------------------------------------------------------------------
-------
Model Name:  ExtraTreesClassifier()
Confusion matrix :
 [[ 124  138]
 [  53 7323]]
Outcome values :
 124 138 53 7323
Classification report :
               precision    recall  f1-score   support

           1       0.70      0.47      0.56       262
           0       0.98      0.99      0.99      7376

    accuracy                           0.97      7638
   macro avg       0.84      0.73      0.78      7638
weighted avg       0.97      0.97      0.97      7638

Accuracy : 97.5 %
Precision : 70.1 %
Recall : 47.3 %
F1 Score : 0.565
Specificity or True Negative Rate : 99.3 %
Balanced Accuracy : 73.3 %
MCC : 0.564
roc_auc_score: 0.733
```



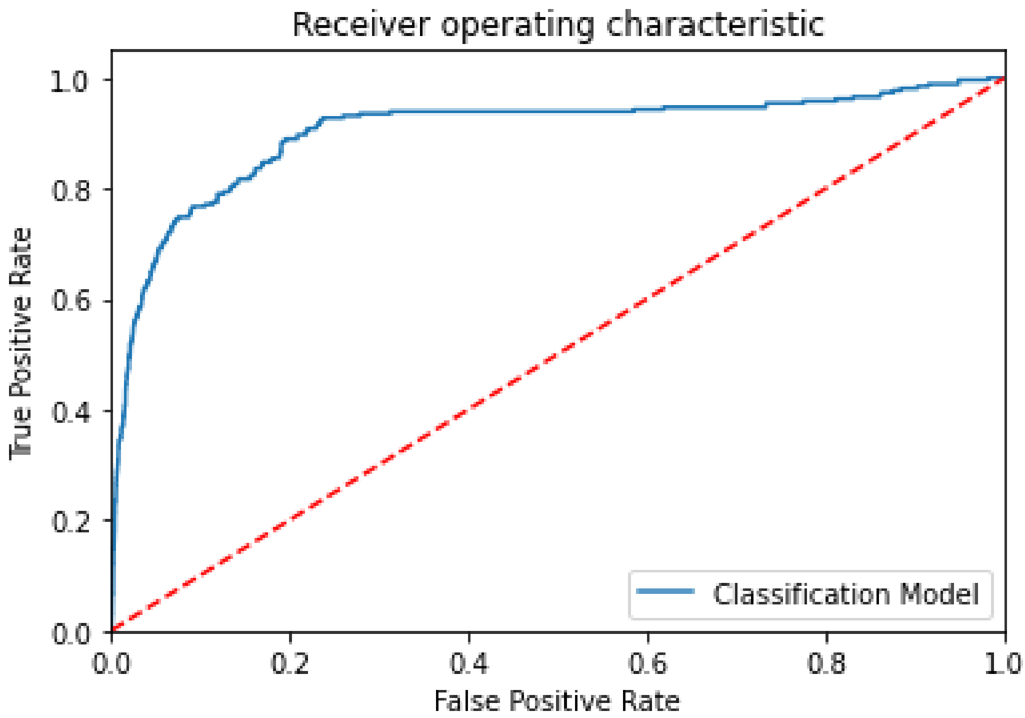

```
---------------------------------------------------------------------------------------------------
-------
Model Name:  KNeighborsClassifier(n_neighbors=1)
Confusion matrix :
 [[ 140  122]
 [ 102 7274]]
Outcome values :
 140 122 102 7274
Classification report :
               precision    recall  f1-score   support

           1       0.58      0.53      0.56       262
           0       0.98      0.99      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.78      0.76      0.77      7638
weighted avg       0.97      0.97      0.97      7638

Accuracy : 97.1 %
Precision : 57.9 %
Recall : 53.4 %
F1 Score : 0.556
Specificity or True Negative Rate : 98.6 %
Balanced Accuracy : 76.0 %
MCC : 0.541
roc_auc_score: 0.76
```




```
---------------------------------------------------------------------------------------------
-------
Model Name:  GaussianNB()
Confusion matrix :
 [[ 262    0]
 [2131 5245]]
Outcome values :
 262 0 2131 5245
Classification report :
               precision    recall  f1-score   support

           1       0.11      1.00      0.20       262
           0       1.00      0.71      0.83      7376

    accuracy                           0.72      7638
   macro avg       0.55      0.86      0.51      7638
weighted avg       0.97      0.72      0.81      7638

Accuracy : 72.1 %
Precision : 10.9 %
Recall : 100.0 %
F1 Score : 0.197
Specificity or True Negative Rate : 71.1 %
Balanced Accuracy : 85.5 %
MCC : 0.279
roc_auc_score: 0.856
```




```
----------------------------------------------------------------------------------------------
-------
Model Name:  SVC(probability=True)
Confusion matrix :
 [[   0  262]
 [   0 7376]]
Outcome values :
 0 262 0 7376
Classification report :
               precision    recall  f1-score   support

           1       0.00      0.00      0.00       262
           0       0.97      1.00      0.98      7376

    accuracy                           0.97      7638
   macro avg       0.48      0.50      0.49      7638
weighted avg       0.93      0.97      0.95      7638

Accuracy : 96.6 %
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
```



RESULTS

In [87]:

```
# display the EMResults
EMResults.head(10)
```

Out[87]:

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
0	LogisticRegression()	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
1	DecisionTreeClassifier()	165	97	197	7179	0.962	0.456	0.63	0.529	0.973	0.517	0.801531	0.802
2	(DecisionTreeClassifier(max_features='auto', r...	148	114	42	7334	0.98	0.779	0.565	0.655	0.994	0.653	0.779596	0.78
3	(ExtraTreeClassifier(random_state=182236865), ...	126	136	58	7318	0.975	0.685	0.481	0.565	0.992	0.562	0.736526	0.736
4	KNeighborsClassifier(n_neighbors=8)	50	212	22	7354	0.969	0.694	0.191	0.299	0.997	0.354	0.593929	0.594
5	GaussianNB()	262	0	2131	5245	0.721	0.109	1.0	0.197	0.711	0.279	0.855545	0.855

	Model Name	True_Positive	False_Negative	False_Positive	True_Negative	Accuracy	Precision	Recall	F1 Score	Specificity	MCC	ROC_AUC_Score	Balanced Accuracy
6	SVC(probability=True)	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
7	LogisticRegression()	0	262	0	7376	0.966	NaN	0.0	0.0	1.0	NaN	0.5	0.5
8	DecisionTreeClassifier()	163	99	217	7159	0.959	0.429	0.622	0.508	0.971	0.496	0.796359	0.796
9	(DecisionTreeClassifier(max_features='auto', r...	141	121	33	7343	0.98	0.81	0.538	0.647	0.996	0.651	0.766847	0.767

PREDICATION OF ALGORITHM

In [88]:

```
#predict the values with knn algorithm
y_predKNN = ModelRF.predict(x_test)
```

In [89]:

```
#create data frame with actual vs predict values
# display the final results

Results=pd.DataFrame({'is_preparatory_A':y_test,'is_preparatory_P':y_pred})

# Merge two dataframes on the index of both the dataframe

ResultsFinal=data_bk2.merge(Results,left_index=True,right_index=True)


# display 5 records randomly

ResultsFinal.sample(5)
```

Out[89]:

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory	is_preparatory_A	is_preparatory_P
7500	7499	2020	IIT	6	AI	Gender-Neutral	IIT-Kharagpur	Mining Engineering	4 Years	B.Tech	GEN	6040	8152	0	0	0
52	53	2016	IIT	6	AI	Gender-Neutral	IIT-Bombay	Engineering Physics	4 Years	B.Tech	SC	360	763	0	0	0

	id	year	institute_type	round_no	quota	pool	institute_short	program_name	program_duration	degree_short	category	opening_rank	closing_rank	is_preparatory	is_preparatory_A	is_preparatory_P
24567	30249	2021	NIT	1	OS	Female-Only	NIT-Kurukshetra	Civil Engineering	4 Years	B.Tech	GEN-EWS	5133	5919	0	0	0
11742	11743	2021	IIT	1	AI	Gender-Neutral	IIT-(BHU) Varanasi	Mining Engineering	5 Years	B.Tech + M.Tech (IDD)	OBC-NCL	4667	4892	0	0	0
21096	25807	2020	NIT	6	OS	Gender-Neutral	NIT-Rourkela	Metallurgical and Materials Engineering	4 Years	B.Tech	GEN-EWS	4341	4908	0	0	0

In [ ]:

: