**Experiment 11:**

**Aim**:

To implement a Naive Bayes classifier for a sample training dataset stored as a .csv file and compute the accuracy using a few test datasets.

**Algorithm**:

1. Load the dataset from the .csv file and prepare the training and test data.
2. Split the dataset into training and test sets using the train\_test\_split function.
3. Train the Naive Bayes classifier on the training data.
4. Predict the outcomes for the test data and compute the accuracy.
5. Evaluate the model's performance using confusion matrix, F1 score, and ROC curve.

**Program**:

python

Copy code

import csv

import pandas as pd

import numpy as np

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import confusion\_matrix, f1\_score, roc\_curve, auc

import matplotlib.pyplot as plt

from itertools import cycle

from scipy import interpolate

import warnings

import random

import math

# Load and preprocess the data

with open('D:\\ML project\\heartdisease.txt', 'r') as in\_file:

stripped = (line.strip() for line in in\_file)

lines = (line.split(",") for line in stripped if line)

with open('heartdisease.csv', 'w', newline='') as out\_file:

writer = csv.writer(out\_file)

writer.writerow(('age', 'sex', 'cp', 'restbp', 'chol', 'fbs', 'restecg',

'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'num'))

writer.writerows(lines)

# Suppress warnings

warnings.filterwarnings("ignore")

# Load the dataset

df = pd.read\_csv('heartdisease.csv', header=None)

training\_x = df.iloc[1:df.shape[0], 0:13]

training\_y = df.iloc[1:df.shape[0], 13:14]

print(df)

# Convert data to numpy arrays

x = np.array(training\_x)

y = np.array(training\_y)

# Split, train, and evaluate the model

for z in range(5):

print("\n\n\nTest Train Split no. ", z+1, "\n\n\n")

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25, random\_state=None)

gnb = GaussianNB()

gnb.fit(x\_train, y\_train.ravel())

y\_pred = gnb.predict(x\_test)

print("\n\nGaussian Naive Bayes model accuracy(in %):", metrics.accuracy\_score(y\_test, y\_pred) \* 100)

# Evaluate using confusion matrix and F1 score

y1 = y\_test.ravel()

y\_pred1 = y\_pred.ravel()

print("\n\n\n\nConfusion Matrix")

cf\_matrix = confusion\_matrix(y1, y\_pred1)

print(cf\_matrix)

print("\n\n\n\nF1 Score")

f\_score = f1\_score(y1, y\_pred1, average='weighted')

print(f\_score)

# Prepare data for ROC curve

y2 = np.zeros(shape=(len(y1), 5))

y3 = np.zeros(shape=(len(y\_pred1), 5))

for i in range(len(y1)):

y2[i][int(y1[i])] = 1

for i in range(len(y\_pred1)):

y3[i][int(y\_pred1[i])] = 1

n\_classes = 5

fpr = dict()

tpr = dict()

roc\_auc = dict()

for i in range(n\_classes):

fpr[i], tpr[i], \_ = roc\_curve(y2[:, i], y3[:, i])

roc\_auc[i] = auc(fpr[i], tpr[i])

fpr["micro"], tpr["micro"], \_ = roc\_curve(y2.ravel(), y3.ravel())

roc\_auc["micro"] = auc(fpr["micro"], tpr["micro"])

lw = 2

all\_fpr = np.unique(np.concatenate([fpr[i] for i in range(n\_classes)]))

mean\_tpr = np.zeros\_like(all\_fpr)

for i in range(n\_classes):

f = interpolate.interp1d(fpr[i], tpr[i], bounds\_error=False, fill\_value='extrapolate')

mean\_tpr += f(all\_fpr)

mean\_tpr /= n\_classes

fpr["macro"] = all\_fpr

tpr["macro"] = mean\_tpr

roc\_auc["macro"] = auc(fpr["macro"], tpr["macro"])

plt.figure()

plt.plot(fpr["micro"], tpr["micro"],

label='micro-average (area = {0:0.2f})'.format(roc\_auc["micro"]),

color='deeppink', linestyle=':', linewidth=4)

plt.plot(fpr["macro"], tpr["macro"],

label='macro-average (area = {0:0.2f})'.format(roc\_auc["macro"]),

color='navy', linestyle=':', linewidth=4)

colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'black'])

for i, color in zip(range(n\_classes), colors):

plt.plot(fpr[i], tpr[i], color=color, lw=lw,

label='ROC of class {0} (area = {1:0.2f})'.format(i, roc\_auc[i]))

plt.plot([0, 1], [0, 1], 'k--', lw=lw)

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 for multi-class')

plt.legend(loc="lower right")

plt.show()

**Output**:



