EX.NO: 3 IMPLEMENT NAIVE BAYES

Aim:

The aim of the Naïve Bayes algorithm is to classify a given set of data points into different classes based on the probability of each data point belonging to a particular class. This algorithm is based on the Bayes theorem, which states that the probability of an event occurring given the prior knowledge of another event can be calculated using conditional probability.

Program:

```
# Importing library
import  math import
random importcsv


# the categorical class names are changed to numberic data#
eg: yes and no encoded to 1 and 0 defencode_class(mydata):
        classes = []
        for i in range(len(mydata)):
                ifmydata[i][-1] not in classes:
                        classes.append(mydata[i][-1])
        for i in range(len(classes)):
                for j in range(len(mydata)):
                        ifmydata[j][-1] == classes[i]:
                                mydata[j][-1] = i
        returnmydata


# Splitting the data
def splitting(mydata, ratio):
        train_num = int(len(mydata) * ratio)
        train = []
        # initiallytestset will have all the dataset
        test = list(mydata)
        whilelen(train) <train_num:
                # index generated randomly from range 0#
                to length of testset
                index = random.randrange(len(test))
                # from testset, pop data rows and put it in train
                train.append(test.pop(index))
        return train, test


# Group the data rows under each class yes or#
no in dictionary eg: dict[yes] and dict[no]
defgroupUnderClass(mydata):
        dict = {}
        for i in range(len(mydata)):
                if (mydata[i][-1] not in dict):
                        dict[mydata[i][-1]] = []
```

```
                dict[mydata[i][-1]].append(mydata[i])
        returndict
# Calculating Meandef
mean(numbers):
        return sum(numbers) / float(len(numbers))

# Calculating Standard Deviation


       defstd_dev(numbers):
        avg = mean(numbers)
        variance = sum([pow(x - avg, 2) for x in numbers]) / float(len(numbers) - 1)
        returnmath.sqrt(variance)

defMeanAndStdDev(mydata):
        info = [(mean(attribute), std_dev(attribute)) for attribute in zip(*mydata)]#
        eg: list = [ [a, b, c], [m, n, o], [x, y, z]]
        # here mean of 1st attribute =(a + m+x), mean of 2nd attribute = (b + n+y)/3#
        delete summaries of last class
        del info[-1]
        return info

# find Mean and Standard Deviation under each class
defMeanAndStdDevForClass(mydata):
        info = {}
        dict = groupUnderClass(mydata)
        forclassValue, instances in dict.items():
                info[classValue] = MeanAndStdDev(instances)
        return info


# Calculate Gaussian Probability Density Function
defcalculateGaussianProbability(x, mean, stdev):
        expo = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
        return (1 / (math.sqrt(2 * math.pi) * stdev)) * expo


# Calculate Class Probabilities
defcalculateClassProbabilities(info, test):
        probabilities = {}
        forclassValue, classSummaries in info.items():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, std_dev = classSummaries[i]x
                        = test[i]
                        probabilities[classValue] *= calculateGaussianProbability(x, mean,
std_dev)
        return probabilities
```

```
# Make prediction - highest probability is the prediction
def predict(info, test):
        probabilities = calculateClassProbabilities(info, test)
        bestLabel, bestProb = None, -1
        forclassValue, probability in probabilities.items():
                ifbestLabel is None or probability >bestProb:
                        bestProb = probability
                        bestLabel = classValue


        returnbestLabel
# returns predictions for a set of examples
defgetPredictions(info, test):
        predictions = []
        for i in range(len(test)):
                result = predict(info, test[i])
                predictions.append(result)
        return predictions

# Accuracy score
defaccuracy_rate(test, predictions):
        correct = 0
        for i in range(len(test)):
                if test[i][-1] == predictions[i]:
                        correct += 1
        return (correct / float(len(test))) * 100.0
# driver code

# add the data path in your system
filename = r'E:\user\MACHINE LEARNING\machine learning algos\Naive
bayes\filedata.csv'


# load the file and store it in mydata list
mydata = csv.reader(open(filename, "rt"))
mydata = list(mydata)
mydata = encode_class(mydata)
for i in range(len(mydata)):
        mydata[i] = [float(x) for x in mydata[i]]
# split ratio = 0.7
# 70% of data is training data and 30% is test data used for testing
ratio = 0.7
train_data, test_data = splitting(mydata, ratio)
print('Total number of examples are: ', len(mydata))
print('Out of these, training examples are: ', len(train_data))
print("Test examples are: ", len(test_data))
```

```
# prepare model
info = MeanAndStdDevForClass(train_data)

# test model
predictions = getPredictions(info, test_data)
accuracy = accuracy_rate(test_data, predictions)
print("Accuracy of your model is: ", accuracy)
```

Output:

```
Total number of examples are: 200
Out of these, training examples are: 140
Test examples are: 60
Accuracy of your model is: 71.2376788
```