Program 5: Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

It is a classification technique based on Bayes? Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as „Naive?.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:

1) Handling Of Data: • load the data from the CSV file and split in to training and test data set. • Training data set can be used to by Naïve Bayes to make predictions. • And Test data set can be used to evaluate the accuracy of the model.

2) Summarize Data: The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value.

• These are required when making predictions to calculate the probability of specific attribute values belonging to each class value.

• summary data can be break down into the following sub-tasks:

a) Separate Data By Class: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.

b) Calculate Mean: We need to calculate the mean of each attribute for a class value. The mean is the central middle or central tendency of the data, and we will use it as the middle of our Gaussian distribution when calculating probabilities.

3) Calculate Standard Deviation: We also need to calculate the standard deviation of each attribute for a class value. The standard deviation describes the variation of spread of the data, and we will use it to characterize the expected spread of each attribute in our Gaussian distribution when calculating probabilities.

4) Summarize Dataset: For a given list of instances (for a class value) we can calculate the mean and the standard deviation for each attribute.

The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.

5) Summarize Attributes By Class: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

3) Make Predictions:

• Making predictions involves calculating the probability that a given data instance belongs to each class,

• then selecting the class with the largest probability as the prediction.

• Finally, estimation of the accuracy of the model by making predictions for each data instance in the test dataset.

4) Evaluate Accuracy: The predictions can be compared to the class values in the test dataset and a classification\ accuracy can be calculated as an accuracy ratio between 0& and 100%.

```
print("\nNaive Bayes Classifier for concept learning problem")
import csv
#import random
import math
#import operator
def safe_div(x,y):
    if y == 0:
        return 0
    return x / y

def loadCsv(filename):
        lines = csv.reader(open(filename))
        dataset = list(lines)
        for i in range(len(dataset)):
                dataset[i] = [float(x) for x in dataset[i]]
        return dataset

def splitDataset(dataset, splitRatio):
        trainSize = int(len(dataset) * splitRatio)
        trainSet = []
        copy = list(dataset)
        i=0
        while len(trainSet) < trainSize:
                #index = random.randrange(len(copy))

                trainSet.append(copy.pop(i))
        return [trainSet, copy]

def separateByClass(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    #print(separated)
    return separated
```

```
def mean(numbers):
        return safe_div(sum(numbers),float(len(numbers)))

def stdev(numbers):
        avg = mean(numbers)
        variance = safe_div(sum([pow(x-avg,2) for x in numbers]),float(len(numbers)-1))
        return math.sqrt(variance)

def summarize(dataset):
   #for attribute in zip(*dataset):
      #print(attribute)
   summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
   del summaries[-1]
   return summaries

def summarizeByClass(dataset):
   separated = separateByClass(dataset)
   summaries = {}
   #p=separated.items();
   #print(p)
   for classValue, instances in separated.items():
     # print(classValue)
      #print(instances)
      summaries[classValue] = summarize(instances)
   #print(summaries)
   return summaries

def calculateProbability(x, mean, stdev):
        exponent = math.exp(-safe_div(math.pow(x-mean,2),(2*math.pow(stdev,2))))
        final = safe_div(1 , (math.sqrt(2*math.pi) * stdev)) * exponent
        return final

def calculateClassProbabilities(summaries, inputVector):
        probabilities = {}
        for classValue, classSummaries in summaries.items():
                probabilities[classValue] = 1
                for i in range(len(classSummaries)):
                        mean, stdev = classSummaries[i]
                        x = inputVector[i]
                        probabilities[classValue] *= calculateProbability(x, mean, stdev)
        return probabilities

def predict(summaries, inputVector):
        probabilities = calculateClassProbabilities(summaries, inputVector)
        bestLabel, bestProb = None, -1
        for classValue, probability in probabilities.items():
                if bestLabel is None or probability > bestProb:
                        bestProb = probability
                        bestLabel = classValue
```

```
        return bestLabel

def getPredictions(summaries, testSet):
        predictions = []
        for i in range(len(testSet)):
                result = predict(summaries, testSet[i])
                predictions.append(result)
        return predictions

def getAccuracy(testSet, predictions):
        correct = 0
        for i in range(len(testSet)):
                if testSet[i][-1] == predictions[i]:
                        correct += 1
        accuracy = safe_div(correct,float(len(testSet))) * 100.0
        return accuracy

def main():
        filename = 'ConceptLearning.csv'
        splitRatio = 0.75
        dataset = loadCsv(filename)
        trainingSet, testSet = splitDataset(dataset, splitRatio)
        print('Split {0} rows into'.format(len(dataset)))

        print('Number of Training data: ' + (repr(len(trainingSet))))
        print('Number of Test Data: ' + (repr(len(testSet))))
        print("\nThe values assumed for the concept learning attributes are\n")
        print("OUTLOOK=> Sunny=1 Overcast=2 Rain=3\nTEMPERATURE=> Hot=1
Mild=2 Cool=3\nHUMIDITY=> High=1 Normal=2\nWIND=> Weak=1 Strong=2")
        print("TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5")
        print("\nThe Training set are:")
        for x in trainingSet:
                print(x)
        print("\nThe Test data set are:")
        for x in testSet:
                print(x)
        print("\n")
        # prepare model
        summaries = summarizeByClass(trainingSet)
        # test model
        predictions = getPredictions(summaries, testSet)
        actual = []
        for i in range(len(testSet)):
                vector = testSet[i]
                actual.append(vector[-1])
        # Since there are five attribute values, each attribute constitutes to 20% accuracy. So if
all attributes match with predictions then 100% accuracy
        print('Actual values: {0}%'.format(actual))
        print('Predictions: {0}%'.format(predictions))
        accuracy = getAccuracy(testSet, predictions)
```

```
        print('Accuracy: {0}%'.format(accuracy))

main()
```

OUTPUT

```
Split 6 rows into
Number of Training data: 4
Number of Test Data: 2
The values assumed for the concept learning attributes are
OUTLOOK=> Sunny=1 Overcast=2 Rain=3
TEMPERATURE=> Hot=1 Mild=2 Cool=3
HUMIDITY=> High=1 Normal=2
WIND=> Weak=1 Strong=2
TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5
The Training set are:
[1.0, 1.0, 1.0, 1.0, 5.0]
[1.0, 1.0, 1.0, 2.0, 5.0]
[2.0, 1.0, 1.0, 2.0, 10.0]
[3.0, 2.0, 1.0, 1.0, 10.0]
The Test data set are:
[3.0, 3.0, 2.0, 1.0, 10.0]
[3.0, 3.0, 2.0, 2.0, 5.0]
Actual values: [10.0]%
Actual values: [10.0, 5.0]%
Predictions: [5.0, 5.0]%
Accuracy: 50.0%
```

Program 6: Assuming a set of documents that need to be classified, use the naïve Bayesian Classifier model to perform this task. Built-in Java classes/API can be used to write the program. Calculate the accuracy, precision, and recall for your data set

```
rom sklearn.datasets import fetch_20newsgroups
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#pprint(fetch_20newsgroups)
categories = ['alt.atheism', 'soc.religion.christian','comp.graphics', 'sci.med']
twenty_train = fetch_20newsgroups(subset='train',categories=categories,shuffle=True)
#pprint(twenty_train)
twenty_test = fetch_20newsgroups(subset='test',categories=categories,shuffle=True)
print(len(twenty_train.data))
print(len(twenty_test.data))
print(twenty_train.target_names)
print("\n".join(twenty_train.data[0].split("\n")))
print(twenty_train.target[0])
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer()
X_train_tf = count_vect.fit_transform(twenty_train.data)
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_tf)
print(X_train_tfidf)
X_train_tfidf.shape
print(X_train_tfidf.shape)
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics
mod = MultinomialNB()
mod.fit(X_train_tfidf, twenty_train.target)
X_test_tf = count_vect.transform(twenty_test.data)
X_test_tfidf = tfidf_transformer.transform(X_test_tf)
predicted = mod.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(twenty_test.target, predicted))
print(classification_report(twenty_test.target,predicted,target_names=twenty_test.target_names
))
#pprint(predicted)

print("confusion matrix is \n",metrics.confusion_matrix(twenty_test.target, predicted))
```

OUTPUT

```
2257
1502
```