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.

Bayes' Theorem is stated as:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where,
P(h|D) is the probability of hypothesis h given the data D. This is called the **posterior probability**.
P(D|h) is the probability of data d given that the hypothesis h was true.
P(h) is the probability of hypothesis h being true. This is called the **prior probability of h.**
P(D) is the probability of the data. This is called the **prior probability of D**

After calculating the posterior probability for a number of different hypotheses h, and is interested in finding the most probable hypothesis h ∈ H given the observed data D. Any such maximally probable hypothesis is called a ***maximum a posteriori (MAP) hypothesis***.

Bayes theorem to calculate the posterior probability of each candidate hypothesis is h_{MAP} is a MAP hypothesis provided

$$h_{MAP} = \arg\max_{h \in H} P(h|D)$$

$$= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)}$$

$$= \arg\max_{h \in H} P(D|h)P(h)$$

(Ignoring P(D) since it is a constant)

Gaussian Naive Bayes

A Gaussian Naive Bayes algorithm is a special type of Naïve Bayes algorithm. It's specifically used when the features have continuous values. It's also assumed that all the features are following a Gaussian distribution i.e., normal distribution

Representation for Gaussian Naive Bayes

We calculate the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

Gaussian Naive Bayes Model from Data

The probability density function for the normal distribution is defined by two parameters (mean and standard deviation) and calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$ Mean

$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu)^2\right]^{0.5}$$ Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$ Normal distribution

Example: Refer the link

http://chem-eng.utoronto.ca/~datamining/dmc/naive_bayesian.htm

Examples:

- The data set used in this program is the ***Pima Indians Diabetes problem***.
- This data set is comprised of 768 observations of medical details for Pima Indians patents. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.
- The attributes are ***Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabeticPedigreeFunction, Age, Outcome***
- Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0)

Sample Examples:

Examples	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetic Pedigree Function	Age	Outcome
1	6	148	72	35	0	33.6	0.627	50	1
2	1	85	66	29	0	26.6	0.351	31	0
3	8	183	64	0	0	23.3	0.672	32	1
4	1	89	66	23	94	28.1	0.167	21	0
5	0	137	40	35	168	43.1	2.288	33	1
6	5	116	74	0	0	25.6	0.201	30	0
7	3	78	50	32	88	31	0.248	26	1
8	10	115	0	0	0	35.3	0.134	29	0
9	2	197	70	45	543	30.5	0.158	53	1
10	8	125	96	0	0	0	0.232	54	1

Program:

```
import csv
import random
import math

def loadcsv(filename):
     lines = csv.reader(open(filename, "r"));
     dataset = list(lines)
     for i in range(len(dataset)):
        #converting strings into numbers for processing
          dataset[i] = [float(x) for x in dataset[i]]

     return dataset

def splitdataset(dataset, splitratio):
    #67% training size
     trainsize = int(len(dataset) * splitratio);
     trainset = []
     copy = list(dataset);
     while len(trainset) < trainsize:
#generate indices for the dataset list randomly to pick ele for
training data
          index = random.randrange(len(copy));
          trainset.append(copy.pop(index))
     return [trainset, copy]

def separatebyclass(dataset):
     separated = {} #dictionary of classes 1 and 0
#creates a dictionary of classes 1 and 0 where the values are
#the instances belonging to each class
     for i in range(len(dataset)):
          vector = dataset[i]
          if (vector[-1] not in separated):
               separated[vector[-1]] = []
          separated[vector[-1]].append(vector)
     return separated

def mean(numbers):
     return sum(numbers)/float(len(numbers))

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

```
def summarize(dataset): #creates a dictionary of classes
     summaries   =   [(mean(attribute),   stdev(attribute))   for
attribute in zip(*dataset)];
     del summaries[-1] #excluding labels +ve or -ve
     return summaries

def summarizebyclass(dataset):
     separated = separatebyclass(dataset);
     #print(separated)
     summaries = {}
     for classvalue, instances in separated.items():
#for key,value in dic.items()
#summaries is a dic of tuples(mean,std) for each class value
          summaries[classvalue] = summarize(instances)
#summarize is used to cal to mean and std
     return summaries

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

def calculateclassprobabilities(summaries, inputvector):
# probabilities contains the all prob of all class of test data
     probabilities = {}
     for classvalue, classsummaries in summaries.items():
#class and attribute information as mean and sd
          probabilities[classvalue] = 1
          for i in range(len(classsummaries)):
               mean, stdev = classsummaries[i] #take mean and
sd of every attribute for class 0 and 1 seperaely
               x = inputvector[i] #testvector's first attribute
               probabilities[classvalue] *=
calculateprobability(x, mean, stdev);#use normal dist
     return probabilities

def predict(summaries, inputvector): #training and test data
is passed
     probabilities = calculateclassprobabilities(summaries,
inputvector)
     bestLabel, bestProb = None, -1
     for classvalue, probability in probabilities.items():
#assigns that class which has the highest prob
          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
     return (correct/float(len(testset))) * 100.0

def main():
     filename = 'naivedata.csv'
     splitratio = 0.67
     dataset = loadcsv(filename);

     trainingset, testset = splitdataset(dataset, splitratio)
     print('Split {0} rows into train={1} and test={2}
rows'.format(len(dataset), len(trainingset), len(testset)))
     # prepare model
     summaries = summarizebyclass(trainingset);
     #print(summaries)
    # test model
     predictions = getpredictions(summaries, testset) #find the
predictions of test data with the training data
     accuracy = getaccuracy(testset, predictions)
     print('Accuracy of the classifier is :
{0}%'.format(accuracy))

main()
```

Output:

Split 768 rows into train=514 and test=254 rows

Accuracy of the classifier is : 71.65354330708661%