Intelligent Systems Lab

Lab No- 5

Name- Adhyyan Tripathi

Roll no -8

Sec C

Reg no – 201700403

Q. 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.

Ans.

1. DIGITS DATASET

```
from sklearn.datasets import
load_digits dataset = load_digits()

X = dataset.data


y = dataset.target
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.4, random_state=1)

from sklearn.naive_bayes import
GaussianNB gnb = GaussianNB()

gnb.fit(X_train, y_train)


y_pred = gnb.predict(X_test)
from sklearn.metrics import accuracy_score

print("Gaussian Naive Bayes accuracy:",
accuracy_score(y_test, y_pred)*100)
```

Output :-

```
Gaussian Naive Bayes accuracy: 83.03198887343532
```

2. Pima indians diabetes

```
import csv
import random
import math
# 1.Data Handling
# 1.1 Loading the Data from csv file of Pima indians diabetes
dataset. def loadcsv(filename):
   lines = csv.reader(open(filename,"r"))
   dataset = list(lines)
   for i in range(len(dataset)):
      # converting the attributes from string to floating point numbers
      dataset[i] = [float(x) for x in dataset[i]]
   return dataset
#1.2 Splitting the Data set into Training Set
def splitDataset(dataset, splitRatio):
   trainSize = int(len(dataset) * splitRatio)
   trainSet = []
   copy = list(dataset)
   while len(trainSet) < trainSize:
      index = random.randrange(len(copy)) # random index
      trainSet.append(copy.pop(index)
return [trainSet, copy]

#2.Summarize Data
#The naive bayes model is comprised of a
#summary of the data in the training dataset.
#This summary is then used when making predictions.
#involves the mean and the standard deviation for each attribute, by class value

#2.1: Separate Data By Class
#Function to categorize the dataset in terms of classes
#The function assumes that the last attribute (-1) is the class value.
#The function returns a map of class values to lists of data
instances. 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)
    return separated
#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

#2.2 : Calculate Mean
def mean(numbers):
    return sum(numbers)/float(len(numbers))
#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.

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

#2.4 : Summarize Dataset
#Summarize Data Set for a list of instances (for a class value)
#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.
def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*dataset)]
    del summaries[-1]
    return summaries
```

```
#2.5 : 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.

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


#3.Make Prediction

#3.1 Calculate Probaility Density Function
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

#3.2 Calculate Class Probabilities
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

#3.3 Prediction : look for the largest probability and return the associated class
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

#4.Make Predictions
# Function which return predictions for list of predictions
# For each instan
def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

#5. Computing Accuracy
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
#Main Function
def main():
    filename = 'C:\\Users\\ADHyyAN\\Desktop\\pima-indians-
    diabetes.csv' splitRatio = 0.67
    dataset = loadcsv(filename)
    #print("\n The Data Set :\n",dataset)
    print("\n The length of the Data Set : ",len(dataset))
    print("\n The Data Set Splitting into Training and Testing \n")
    trainingSet, testSet = splitDataset(dataset, splitRatio)
    print('\n Number of Rows in Training Set:{0} rows'.format(len(trainingSet)))
    print('\n Number of Rows in Testing Set:{0} rows'.format(len(testSet)))
    print("\n First Five Rows of Training Set:\n")
    for i in range(0,5):
```

```
        print(trainingSet[i],"\n")

    print("\n First Five Rows of Testing Set:\n")
    for i in range(0,5):
        print(testSet[i],"\n")
    # prepare model
    summaries = summarizeByClass(trainingSet)
    print("\n Model Summaries:\n",summaries)
    # test model
    predictions = getPredictions(summaries, testSet)
    print("\nPredictions:\n",predictions)
    accuracy = getAccuracy(testSet, predictions)
    print('\n Accuracy: {0}%'.format(accuracy))
main()
```

OUTPUT-

```
The length of the Data Set :  768
The Data Set Splitting into Training and Testing

Number of Rows in Training Set:514 rows
Number of Rows in Testing Set:254 rows
First Five Rows of Training Set:
[4.0, 154.0, 62.0, 31.0, 284.0, 32.8, 0.237, 23.0, 0.0]
[9.0, 91.0, 68.0, 0.0, 0.0, 24.2, 0.2, 58.0, 0.0]
[0.0, 73.0, 0.0, 0.0, 0.0, 21.1, 0.342, 25.0, 0.0]
[2.0, 120.0, 54.0, 0.0, 0.0, 26.8, 0.455, 27.0, 0.0]
[6.0, 154.0, 74.0, 32.0, 193.0, 29.3, 0.839, 39.0, 0.0]

First Five Rows of Testing Set:
[6.0, 148.0, 72.0, 35.0, 0.0, 33.6, 0.627, 50.0, 1.0]
[5.0, 116.0, 74.0, 0.0, 0.0, 25.6, 0.201, 30.0, 0.0]
[4.0, 110.0, 92.0, 0.0, 0.0, 37.6, 0.191, 30.0, 0.0]
[1.0, 189.0, 60.0, 23.0, 846.0, 30.1, 0.398, 59.0, 1.0]
[0.0, 118.0, 84.0, 47.0, 230.0, 45.8, 0.551, 31.0, 1.0]
```

```
Model Summaries:
[(0.0: [(3.4341317365269461, 3.0432571545499534), (109.55389221556887, 26.455840589965778), (68.17365269461078, 18.167218616601758), (19.568862275449103, 15.020498123126961), (66.65269461077844, 94.17490137915908), (30.485928143712595, 7.657131528847889), (0.42223053892215534, 0.2869987247594012), (31.443113772455089, 11.772674009687373)], 1.0: [(4.7222222222222222, 3.581541061459774), (139.75555555555556, 33.728065037858514), (70.94444444444444, 21.213936352021771), (22.122222222222224, 17.475171710717373), (102.68333333333334, 130.95674901177041), (34.703333333333335, 7.1889805494786631), (0.5310388888888894, 0.35140294024405431), (36.805555555555556, 10.566679590426146)]]

Predictions:
[1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0,
0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 1.0,
1.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.
0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0,
0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 0.
0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 0.0]

Accuracy: 77.55905511811024%
```

Accuracy = 77.5590

3. WINE DATASET

```
from sklearn.datasets import load_wine
dataset = load_wine()
X = dataset.data

y = dataset.target
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
0.4, random_state=1)

from sklearn.naive_bayes import
GaussianNB gnb = GaussianNB()

gnb.fit(X_train, y_train)


y_pred = gnb.predict(X_test)

from sklearn.metrics import accuracy_score

print("Gaussian Naive Bayes accuracy:",
accuracy_score(y_test, y_pred)*100)
```

Output :-

```
Gaussian Naive Bayes accuracy: 98.61111111111111
```