***NAME-RASHIKA ARUN***

***REGN0-201700409***

***SEC-C***

***LAB 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.***

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 instance

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***

