**WEB MINING**

**16BCE1186**

**HARSH TRIPATHI**

**NAÏVE BAYES**

**UCI PHISING DATASET**

**The dataset was in (.ARFF) format and was converted to (.CSV) by a converter and then further processing and output are shown below.**

**CODE:**

import csv

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

import random

def splitDataset(dataset, splitRatio):

trainSize = int(len(dataset) \* splitRatio)

trainSet = []

copy = list(dataset)

while len(trainSet) < trainSize:

index = random.randrange(len(copy))

trainSet.append(copy.pop(index))

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)

return separated

import math

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):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]

del summaries[-1]

return summaries

def summarizeByClass(dataset):

separated = separateByClass(dataset)

summaries = {}

for classValue, instances in separated.items():

summaries[classValue] = summarize(instances)

return summaries

def calculateProbability(x, mean, stdev):

if((2\*math.pow(stdev,2))==0):

exponent = math.exp(-(math.pow(x-mean,2)/1))

else:

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2))))

if((math.sqrt(2\*math.pi) \* stdev)!=0):

return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

else:

return(1\*exponent)

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 x in range(len(testSet)):

if testSet[x][-1] == predictions[x]:

correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

splitRatio = 0.67

filename = 'data.csv'

dataset = loadCsv(filename)

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('Split {0} rows into train = {1} and test = {2} rows')

summaries = summarizeByClass(trainingSet)

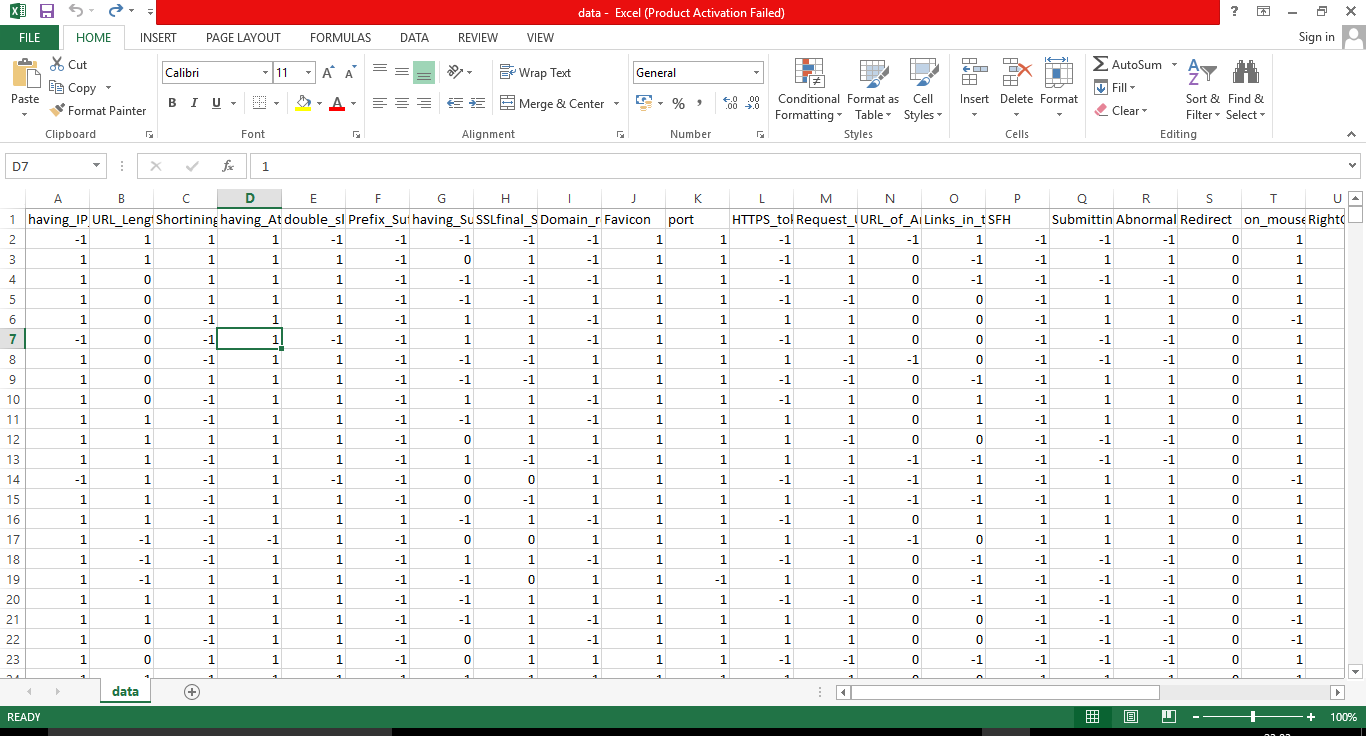
predictions = getPredictions(summaries, testSet)

accuracy = getAccuracy(testSet, predictions)

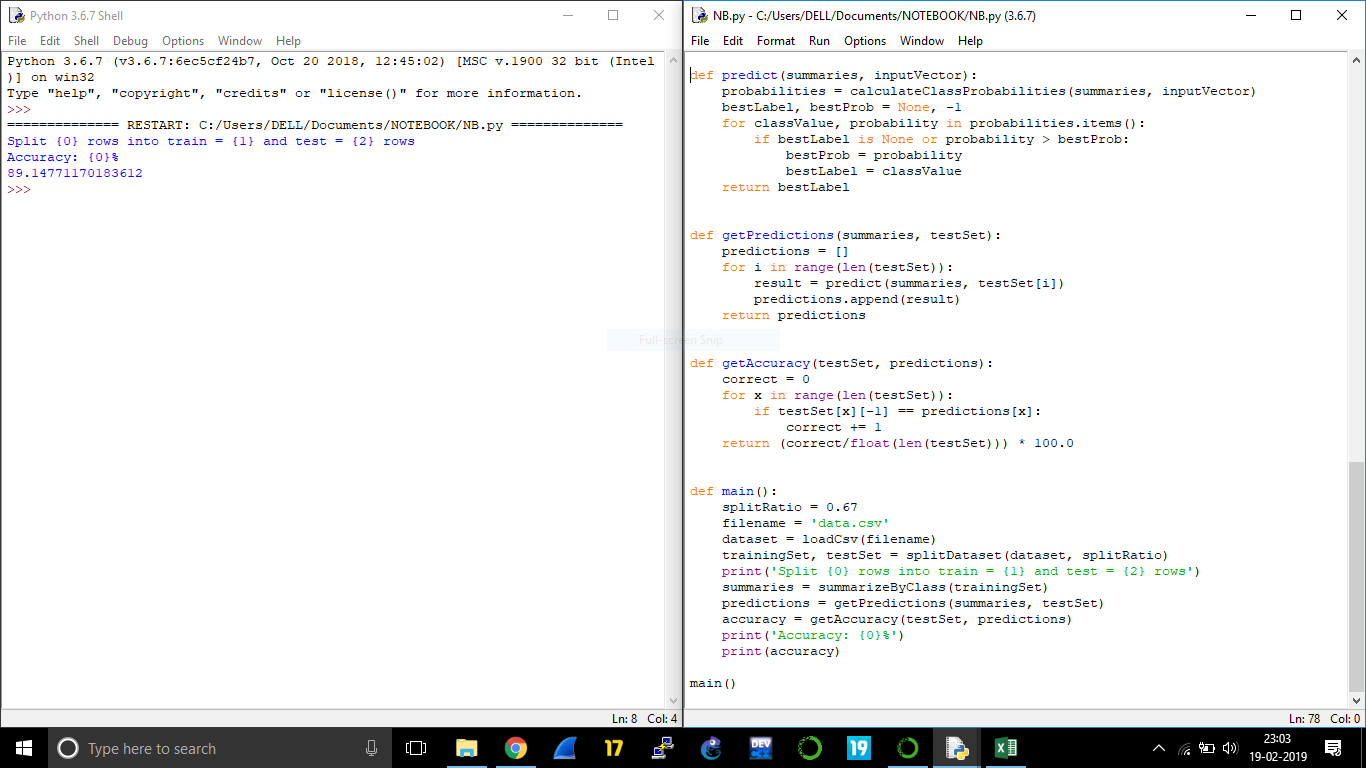
print('Accuracy: {0}%', accuracy)

main()

**DATASET🡪**



**OUTPUT ->**

****