print("\nNaive Bayes Classifier for concept learning problem")

import csv

import math

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)

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):

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):

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 = 'tennis\_naive.csv'

splitRatio = 0.9

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()