

```

"""CSE 6363 - HW01 - Part 3
"""
import math
from math import sqrt
from math import pi
from math import exp
import numpy as np

W = 'W'
M = 'M'
feature = ('height', 'weight', 'age')

trainData_noAge = {((170, 57), W), ((190, 95), M), ((150, 45), W),
((168, 65), M), ((175, 78), M), ((185, 90), M), ((171, 65), W),
((155, 48), W), ((165, 60), W), ((182, 80), M), ((175, 69), W),
((178, 80), M), ((160, 50), W), ((170, 72), M)}

trainData = {((170, 57, 32), W), ((190, 95, 28), M), ((150, 45, 35), W),
((168, 65, 29), M), ((175, 78, 26), M), ((185, 90, 32), M), ((171, 65, 28), W),
((155, 48, 31), W), ((165, 60, 27), W), ((182, 80, 30), M), ((175, 69, 28), W),
((178, 80, 27), M), ((160, 50, 31), W), ((170, 72, 30), M)}

testX = {(162, 53, 28), (168, 75, 32), (175, 70, 30), (180, 85, 29)}
testX_noAge = {(162, 53), (168, 75), (175, 70), (180, 85)}
class naiveBayes:
    def __init__(self, trainXY, testX, precision=4):
        self.precision = precision
        trainXY = np.asarray([sublist for sublist in trainXY], dtype=object)
        testX = np.asarray([sublist for sublist in testX], dtype=object)
        trainYclasses = trainXY[:, -1]
        trainY = list()
        for datum in range(len(trainYclasses)):
            if trainYclasses[datum] == W:
                trainY.append(0)
            else:
                trainY.append(1)

        print()
        print()
        print("trainXY:")
        print(trainXY)

        print()
        print()
        print('testX: ')
        print(testX)

        class_summaries = self.get_all_summaries(trainXY)
        self.print_summaries(class_summaries)
        predictions = list()
        for i in range(len(testX)):
            probs = self.get_class_probability(class_summaries, testX[i])
            print('test datum[' + str(i) + ']: ', testX[i])
            print('probabilities: ', probs)
            if probs[M] > probs[W]:
                print('prediction: M')
                predictions.append(M)
            else:
                print('prediction: W')

```

```

        predictions.append(W)
    print()
    print('test set: ', testX)
    print()
    print("predictions for test set: ", predictions)
    return
    """ end of naiveBayes init """

""" separates training data by class label """
def get_class_summary(self, dataX):
    summary = list()
    """
    print()
    print()
    print('get class summary: ')
    print('dataX: ', dataX)
    print()
    print()
    print('range(len(dataX[0])): ', range(len(dataX[0])))
    print('range(len(dataX)): ', range(len(dataX)))
    """
    for j in range(len(dataX[0])):
        Xi = list()
        for i in range(len(dataX)):
            Xi.append(dataX[i][j])
            summary.append((self.get_mean(Xi),
                            self.get_var(Xi),
                            self.get_stdev(Xi),
                            len(Xi)))
    return summary

def get_all_summaries(self, trainXY):
    XYclasses = self.get_XYclasses(trainXY=trainXY)
    self.print_sep_classes(XYclasses)
    """
    print()
    print('get all class summaries:')
    """
    summaries = dict()
    for label, dataX in XYclasses.items():
        """
        print('label: ', label)
        print('dataX: ', dataX)
        """
        summaries[label] = self.get_class_summary(dataX)
    return summaries

def print_summaries(self, summaries):
    print()
    print()
    print('get all class probabilities:')
    print('summaries[M]: ', summaries[M])
    print('summaries[W]: ', summaries[W])
    return

def get_XYclasses(self, trainXY):
    classes = dict()
    for i in range(len(trainXY)): # row
        datum = trainXY[i] # ith datum/row

```

```

        label = datum[-1]
        if label not in classes:
            classes[label] = list()
        classes[label].append(datum[0])
    return classes

def print_sep_classes(self, classes):
    print()
    print()
    print('print separated classes:')
    print("label: M")
    for i in range(len(classes[M])):
        print(classes[M][i])
    print()
    print("label: W")
    for i in range(len(classes[W])):
        print(classes[W][i])
    return

""" get mean of a list of numbers """
def get_mean(self, numList):
    """
    print()
    print('numList: ', numList)
    """
    mean = sum(numList)/float(len(numList))
    return round(mean, self.precision)

def get_var(self, numList):
    mean = self.get_mean(numList)
    tmp = 0.0
    for i in range(len(numList)):
        tmp += (numList[i] - mean)**2
    var = tmp/float(len(numList))
    return round(var, self.precision)

def get_stdev(self, numList):
    var = self.get_var(numList)
    return round(sqrt(var), self.precision)

def gaussian_PDF(self, x, mean, sd):
    expon = exp(-(((x-mean)**2)/(2*(sd**2))))
    prob_of_x = (1/(sqrt(2*pi)*sd)) * expon
    print('xi:', x, ' | mean:', mean, ' | sd:', sd, ' | p:', round(prob_of_x,
self.precision))
    return round(prob_of_x, self.precision)

''' calculate all class probabilities for datum, given class summaries (trainXY)
'''
def get_class_probability(self, class_summaries, datum):
    print()
    print()
    print()
    print("----->test datum: ", datum)
    summaries = class_summaries
    trainXY_size = 0 # training data size, all classified samples
    probs = dict()
    for label in summaries: # for each class
        trainXY_size += summaries[label][0][3] # samples per class

```

```

    for label, summary in summaries.items():
        print('-----> label: ', label)
        probs[label] = summaries[label][0][3]/float(trainXY_size)
        for i in range(len(summary)):
            print('X['+i+']: ', feature[i])
            mean, var, sd, f = summary[i]
            prob_i = self.gaussian_PDF(datum[i], mean, sd)
            print("P(", feature[i], '|', label, '):', prob_i)
            probs[label] *= prob_i
        print()
    print()
    return probs

""" end of naiveBayes class """

if __name__ == '__main__':
    """ Part a and b """
    naiveBayes(trainData, testX)

    """ Part c """
    naiveBayes(trainData_noAge, testX_noAge)

```