**Program 6: 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.**

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

import csv

import random

import math

import operator

def safe\_div(x,y):

 if y == 0:

  return 0

 return x/y

# 1.Data Handling

# 1.1 Loading the Data from csv file of ConceptLearning dataset.

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

#1.2 Splitting the Data set into Training Set

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]

#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 safe\_div(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 = safe\_div(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)

  print("Summarize Attributes By Class")

  print(summaries)

  print(" ")

  return summaries

#3.Make Prediction

#3.1 Calculate Probaility Density Function

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

#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

  accuracy = safe\_div(correct,float(len(testSet))) \* 100.0

  return accuracy

def main():

  filename = 'ConceptLearning.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()

**output:**

Naive Bayes Classifier for concept learning problem

Split 14 rows into

Number of Training data: 12

Number of Test Data: 2

The values assumed for the concept learning attributes are

OUTLOOK=> Sunny=1 Overcast=2 Rain=3

TEMPERATURE=> Hot=1 Mild=2 Cool=3

HUMIDITY=> High=1 Normal=2

WIND=> Weak=1 Strong=2

TARGET CONCEPT:PLAY TENNIS=> Yes=10 No=5

The Training set are:

[1.0, 1.0, 1.0, 1.0, 5.0]

The Test data set are:

[1.0, 1.0, 1.0, 2.0, 5.0]

The Test data set are:

[2.0, 1.0, 1.0, 1.0, 10.0]

The Test data set are:

[3.0, 2.0, 1.0, 1.0, 10.0]

The Test data set are:

[3.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:

[3.0, 3.0, 2.0, 2.0, 5.0]

The Test data set are:

[2.0, 3.0, 2.0, 2.0, 10.0]

The Test data set are:

[1.0, 2.0, 1.0, 1.0, 5.0]

The Test data set are:

[1.0, 3.0, 2.0, 1.0, 10.0]

The Test data set are:

[3.0, 2.0, 2.0, 1.0, 10.0]

The Test data set are:

[1.0, 2.0, 2.0, 2.0, 10.0]

The Test data set are:

[2.0, 2.0, 1.0, 2.0, 10.0]

The Test data set are:

[2.0, 1.0, 2.0, 1.0, 10.0]

[3.0, 2.0, 1.0, 2.0, 5.0]

Summarize Attributes By Class

{5.0: [(1.5, 1.0), (1.75, 0.9574271077563381), (1.25, 0.5), (1.5, 0.5773502691896257)], 10.0: [(2.125, 0.8345229603962802), (2.25, 0.7071067811865476), (1.625, 0.5175491695067657), (1.375, 0.5175491695067657)]}

Actual values: [5.0]%

Predictions: [5.0, 5.0]%

Accuracy: 50.0%

**Program 7: Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering using k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

import sklearn.metrics as sm

import pandas as pd

import numpy as np

l1 = [0,1,2]

def rename(s):

  l2 = []

  for i in s:

    if i not in l2:

      l2.append(i)

  for i in range(len(s)):

    pos = l2.index(s[i])

    s[i] = l1[pos]

  return s

# import some data to play with

iris = datasets.load\_iris()

print("\n IRIS DATA :",iris.data);

print("\n IRIS FEATURES :\n",iris.feature\_names)

print("\n IRIS TARGET  :\n",iris.target)

print("\n IRIS TARGET NAMES:\n",iris.target\_names)

# Store the inputs as a Pandas Dataframe and set the column names

X = pd.DataFrame(iris.data)

#print(X)

X.columns = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width']

#print(X.columns) #print("X:",x)

#print("Y:",y)

y = pd.DataFrame(iris.target)

y.columns = ['Targets']

# Set the size of the plot

plt.figure(figsize=(14,7))

# Create a colormap

colormap = np.array(['red', 'lime', 'black'])

# Plot Sepal

plt.subplot(1,2,1)

plt.scatter(X.Sepal\_Length,X.Sepal\_Width, c=colormap[y.Targets], s=40)

plt.title('Sepal')

plt.subplot(1,2,2)

plt.scatter(X.Petal\_Length,X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Petal')

plt.show()

print("Actual Target is:\n", iris.target)

# K Means Cluster

model = KMeans(n\_clusters=3)

model.fit(X)

# Set the size of the plot

plt.figure(figsize=(14,7))

# Create a colormap

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications

plt.subplot(1,2,1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y.Targets], s=40)

plt.title('Real Classification')

# Plot the Models Classifications

plt.subplot(1,2,2)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[model.labels\_], s=40)

plt.title('K Mean Classification')

plt.show()

km = rename(model.labels\_)

print("\nWhat KMeans thought: \n", km)

print("Accuracy of KMeans is ",sm.accuracy\_score(y, km))

print("Confusion Matrix for KMeans is \n",sm.confusion\_matrix(y, km))

#The GaussianMixture scikit-learn class can be used to model this problem

#and estimate the parameters of the distributions using the expectation-maximization algorithm.

from sklearn import preprocessing

scaler = preprocessing.StandardScaler()

scaler.fit(X)

xsa = scaler.transform(X)

xs = pd.DataFrame(xsa, columns = X.columns)

print("\n",xs.sample(5))

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3)

gmm.fit(xs)

y\_cluster\_gmm = gmm.predict(xs)

plt.subplot(1, 2, 1)

plt.scatter(X.Petal\_Length, X.Petal\_Width, c=colormap[y\_cluster\_gmm], s=40)

plt.title('GMM Classification')

plt.show()

em = rename(y\_cluster\_gmm)

print("\nWhat EM thought: \n", em)

print("Accuracy of EM is ",sm.accuracy\_score(y, em))

print("Confusion Matrix for EM is \n", sm.confusion\_matrix(y, em))

**output :**

IRIS DATA : [[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1. ]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1. ]

[5.9 3. 4.2 1.5]

[6. 2.2 4. 1. ]

[6.1 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1. ]

[6.2 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

[6.3 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

[6.4 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3. 5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1. ]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1. ]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

[6. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

[5. 2.3 3.3 1. ]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2. ]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5. 2. ]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2. ]

[7.7 2.8 6.7 2. ]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2. ]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2. ]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

IRIS FEATURES :

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

IRIS TARGET :

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

IRIS TARGET NAMES:

['setosa' 'versicolor' 'virginica']

Actual Target is:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

What KMeans thought:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 2 1 2 2 2 2

2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 2 2 2 2 2 1 2 2 2 2 1 2 2 2 1 2 2 2 1 2

2 1]

Accuracy of KMeans is 0.8933333333333333

Confusion Matrix for KMeans is

[[50 0 0]

[ 0 48 2]

[ 0 14 36]]

Sepal\_Length Sepal\_Width Petal\_Length Petal\_Width

67 -0.052506 -0.822570 0.194384 -0.262387

72 0.553333 -1.282963 0.649083 0.395774

117 2.249683 1.709595 1.672157 1.317199

55 -0.173674 -0.592373 0.421734 0.132510

106 -1.143017 -1.282963 0.421734 0.659038

What EM thought:

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 1

1 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

Accuracy of EM is 0.9666666666666667

Confusion Matrix for EM is

[[50 0 0]

[ 0 45 5]

[ 0 0 50]]

**Program 8: Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.**

from sklearn.datasets import load\_iris

from sklearn.neighbors import KNeighborsClassifier

import numpy as np

from sklearn.model\_selection import train\_test\_split

iris\_dataset=load\_iris()

#display the iris dataset

print("\n IRIS FEATURES \ TARGET NAMES: \n ", iris\_dataset.target\_names)

for i in range(len(iris\_dataset.target\_names)):

    print("\n[{0}]:[{1}]".format(i,iris\_dataset.target\_names[i]))

print("\n IRIS DATA :\n",iris\_dataset["data"])

#split the data into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris\_dataset["data"], iris\_dataset["target"], random\_state=0)

print("\n Target :\n",iris\_dataset["target"])

print("\n X TRAIN \n", X\_train)

print("\n X TEST \n", X\_test)

print("\n Y TRAIN \n", y\_train)

print("\n Y TEST \n", y\_test)

#train and fit the model

kn = KNeighborsClassifier(n\_neighbors=5)

kn.fit(X\_train, y\_train)

#predicting from model

x\_new = np.array([[5, 2.9, 1, 0.2]])

print("\n XNEW \n",x\_new)

prediction = kn.predict(x\_new)

print("\n Predicted target value: {}\n".format(prediction))

print("\n Predicted feature name: {}\n".format(iris\_dataset["target\_names"][prediction]))

i=1

x= X\_test[i]

x\_new = np.array([x])

print("\n XNEW \n",x\_new)

for i in range(len(X\_test)):

  x = X\_test[i]

  x\_new = np.array([x])

  prediction = kn.predict(x\_new)

  print("\n Actual : {0} {1}, Predicted :{2}{3}".format(y\_test[i],iris\_dataset["target\_names"][y\_test[i]],prediction,iris\_dataset["target\_names"][ prediction]))

print("\n TEST SCORE[ACCURACY]: {:.2f}\n".format(kn.score(X\_test, y\_test)))

**output:**

IRIS FEATURES \ TARGET NAMES:

['setosa' 'versicolor' 'virginica']

[0]:[setosa]

[1]:[versicolor]

[2]:[virginica]

IRIS DATA :

[[5.1 3.5 1.4 0.2]

[4.9 3. 1.4 0.2]

[4.7 3.2 1.3 0.2]

[4.6 3.1 1.5 0.2]

[5. 3.6 1.4 0.2]

[5.4 3.9 1.7 0.4]

[4.6 3.4 1.4 0.3]

[5. 3.4 1.5 0.2]

[4.4 2.9 1.4 0.2]

[4.9 3.1 1.5 0.1]

[5.4 3.7 1.5 0.2]

[4.8 3.4 1.6 0.2]

[4.8 3. 1.4 0.1]

[4.3 3. 1.1 0.1]

[5.8 4. 1.2 0.2]

[5.7 4.4 1.5 0.4]

[5.4 3.9 1.3 0.4]

[5.1 3.5 1.4 0.3]

[5.7 3.8 1.7 0.3]

[5.1 3.8 1.5 0.3]

[5.4 3.4 1.7 0.2]

[5.1 3.7 1.5 0.4]

[4.6 3.6 1. 0.2]

[5.1 3.3 1.7 0.5]

[4.8 3.4 1.9 0.2]

[5. 3. 1.6 0.2]

[5. 3.4 1.6 0.4]

[5.2 3.5 1.5 0.2]

[5.2 3.4 1.4 0.2]

[4.7 3.2 1.6 0.2]

[4.8 3.1 1.6 0.2]

[5.4 3.4 1.5 0.4]

[5.2 4.1 1.5 0.1]

[5.5 4.2 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5. 3.2 1.2 0.2]

[5.5 3.5 1.3 0.2]

[4.9 3.6 1.4 0.1]

[4.4 3. 1.3 0.2]

[5.1 3.4 1.5 0.2]

[5. 3.5 1.3 0.3]

[4.5 2.3 1.3 0.3]

[4.4 3.2 1.3 0.2]

[5. 3.5 1.6 0.6]

[5.1 3.8 1.9 0.4]

[4.8 3. 1.4 0.3]

[5.1 3.8 1.6 0.2]

[4.6 3.2 1.4 0.2]

[5.3 3.7 1.5 0.2]

[5. 3.3 1.4 0.2]

[7. 3.2 4.7 1.4]

[6.4 3.2 4.5 1.5]

[6.9 3.1 4.9 1.5]

[5.5 2.3 4. 1.3]

[6.5 2.8 4.6 1.5]

[5.7 2.8 4.5 1.3]

[6.3 3.3 4.7 1.6]

[4.9 2.4 3.3 1. ]

[6.6 2.9 4.6 1.3]

[5.2 2.7 3.9 1.4]

[5. 2. 3.5 1. ]

[5.9 3. 4.2 1.5]

[6. 2.2 4. 1. ]

[6.1 2.9 4.7 1.4]

[5.6 2.9 3.6 1.3]

[6.7 3.1 4.4 1.4]

[5.6 3. 4.5 1.5]

[5.8 2.7 4.1 1. ]

[6.2 2.2 4.5 1.5]

[5.6 2.5 3.9 1.1]

[5.9 3.2 4.8 1.8]

[6.1 2.8 4. 1.3]

[6.3 2.5 4.9 1.5]

[6.1 2.8 4.7 1.2]

[6.4 2.9 4.3 1.3]

[6.6 3. 4.4 1.4]

[6.8 2.8 4.8 1.4]

[6.7 3. 5. 1.7]

[6. 2.9 4.5 1.5]

[5.7 2.6 3.5 1. ]

[5.5 2.4 3.8 1.1]

[5.5 2.4 3.7 1. ]

[5.8 2.7 3.9 1.2]

[6. 2.7 5.1 1.6]

[5.4 3. 4.5 1.5]

[6. 3.4 4.5 1.6]

[6.7 3.1 4.7 1.5]

[6.3 2.3 4.4 1.3]

[5.6 3. 4.1 1.3]

[5.5 2.5 4. 1.3]

[5.5 2.6 4.4 1.2]

[6.1 3. 4.6 1.4]

[5.8 2.6 4. 1.2]

[5. 2.3 3.3 1. ]

[5.6 2.7 4.2 1.3]

[5.7 3. 4.2 1.2]

[5.7 2.9 4.2 1.3]

[6.2 2.9 4.3 1.3]

[5.1 2.5 3. 1.1]

[5.7 2.8 4.1 1.3]

[6.3 3.3 6. 2.5]

[5.8 2.7 5.1 1.9]

[7.1 3. 5.9 2.1]

[6.3 2.9 5.6 1.8]

[6.5 3. 5.8 2.2]

[7.6 3. 6.6 2.1]

[4.9 2.5 4.5 1.7]

[7.3 2.9 6.3 1.8]

[6.7 2.5 5.8 1.8]

[7.2 3.6 6.1 2.5]

[6.5 3.2 5.1 2. ]

[6.4 2.7 5.3 1.9]

[6.8 3. 5.5 2.1]

[5.7 2.5 5. 2. ]

[5.8 2.8 5.1 2.4]

[6.4 3.2 5.3 2.3]

[6.5 3. 5.5 1.8]

[7.7 3.8 6.7 2.2]

[7.7 2.6 6.9 2.3]

[6. 2.2 5. 1.5]

[6.9 3.2 5.7 2.3]

[5.6 2.8 4.9 2. ]

[7.7 2.8 6.7 2. ]

[6.3 2.7 4.9 1.8]

[6.7 3.3 5.7 2.1]

[7.2 3.2 6. 1.8]

[6.2 2.8 4.8 1.8]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.1]

[7.2 3. 5.8 1.6]

[7.4 2.8 6.1 1.9]

[7.9 3.8 6.4 2. ]

[6.4 2.8 5.6 2.2]

[6.3 2.8 5.1 1.5]

[6.1 2.6 5.6 1.4]

[7.7 3. 6.1 2.3]

[6.3 3.4 5.6 2.4]

[6.4 3.1 5.5 1.8]

[6. 3. 4.8 1.8]

[6.9 3.1 5.4 2.1]

[6.7 3.1 5.6 2.4]

[6.9 3.1 5.1 2.3]

[5.8 2.7 5.1 1.9]

[6.8 3.2 5.9 2.3]

[6.7 3.3 5.7 2.5]

[6.7 3. 5.2 2.3]

[6.3 2.5 5. 1.9]

[6.5 3. 5.2 2. ]

[6.2 3.4 5.4 2.3]

[5.9 3. 5.1 1.8]]

Target :

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

2 2]

X TRAIN

[[5.9 3. 4.2 1.5]

[5.8 2.6 4. 1.2]

[6.8 3. 5.5 2.1]

[4.7 3.2 1.3 0.2]

[6.9 3.1 5.1 2.3]

[5. 3.5 1.6 0.6]

[5.4 3.7 1.5 0.2]

[5. 2. 3.5 1. ]

[6.5 3. 5.5 1.8]

[6.7 3.3 5.7 2.5]

[6. 2.2 5. 1.5]

[6.7 2.5 5.8 1.8]

[5.6 2.5 3.9 1.1]

[7.7 3. 6.1 2.3]

[6.3 3.3 4.7 1.6]

[5.5 2.4 3.8 1.1]

[6.3 2.7 4.9 1.8]

[6.3 2.8 5.1 1.5]

[4.9 2.5 4.5 1.7]

[6.3 2.5 5. 1.9]

[7. 3.2 4.7 1.4]

[6.5 3. 5.2 2. ]

[6. 3.4 4.5 1.6]

[4.8 3.1 1.6 0.2]

[5.8 2.7 5.1 1.9]

[5.6 2.7 4.2 1.3]

[5.6 2.9 3.6 1.3]

[5.5 2.5 4. 1.3]

[6.1 3. 4.6 1.4]

[7.2 3.2 6. 1.8]

[5.3 3.7 1.5 0.2]

[4.3 3. 1.1 0.1]

[6.4 2.7 5.3 1.9]

[5.7 3. 4.2 1.2]

[5.4 3.4 1.7 0.2]

[5.7 4.4 1.5 0.4]

[6.9 3.1 4.9 1.5]

[4.6 3.1 1.5 0.2]

[5.9 3. 5.1 1.8]

[5.1 2.5 3. 1.1]

[4.6 3.4 1.4 0.3]

[6.2 2.2 4.5 1.5]

[7.2 3.6 6.1 2.5]

[5.7 2.9 4.2 1.3]

[4.8 3. 1.4 0.1]

[7.1 3. 5.9 2.1]

[6.9 3.2 5.7 2.3]

[6.5 3. 5.8 2.2]

[6.4 2.8 5.6 2.1]

[5.1 3.8 1.6 0.2]

[4.8 3.4 1.6 0.2]

[6.5 3.2 5.1 2. ]

[6.7 3.3 5.7 2.1]

[4.5 2.3 1.3 0.3]

[6.2 3.4 5.4 2.3]

[4.9 3. 1.4 0.2]

[5.7 2.5 5. 2. ]

[6.9 3.1 5.4 2.1]

[4.4 3.2 1.3 0.2]

[5. 3.6 1.4 0.2]

[7.2 3. 5.8 1.6]

[5.1 3.5 1.4 0.3]

[4.4 3. 1.3 0.2]

[5.4 3.9 1.7 0.4]

[5.5 2.3 4. 1.3]

[6.8 3.2 5.9 2.3]

[7.6 3. 6.6 2.1]

[5.1 3.5 1.4 0.2]

[4.9 3.1 1.5 0.2]

[5.2 3.4 1.4 0.2]

[5.7 2.8 4.5 1.3]

[6.6 3. 4.4 1.4]

[5. 3.2 1.2 0.2]

[5.1 3.3 1.7 0.5]

[6.4 2.9 4.3 1.3]

[5.4 3.4 1.5 0.4]

[7.7 2.6 6.9 2.3]

[4.9 2.4 3.3 1. ]

[7.9 3.8 6.4 2. ]

[6.7 3.1 4.4 1.4]

[5.2 4.1 1.5 0.1]

[6. 3. 4.8 1.8]

[5.8 4. 1.2 0.2]

[7.7 2.8 6.7 2. ]

[5.1 3.8 1.5 0.3]

[4.7 3.2 1.6 0.2]

[7.4 2.8 6.1 1.9]

[5. 3.3 1.4 0.2]

[6.3 3.4 5.6 2.4]

[5.7 2.8 4.1 1.3]

[5.8 2.7 3.9 1.2]

[5.7 2.6 3.5 1. ]

[6.4 3.2 5.3 2.3]

[6.7 3. 5.2 2.3]

[6.3 2.5 4.9 1.5]

[6.7 3. 5. 1.7]

[5. 3. 1.6 0.2]

[5.5 2.4 3.7 1. ]

[6.7 3.1 5.6 2.4]

[5.8 2.7 5.1 1.9]

[5.1 3.4 1.5 0.2]

[6.6 2.9 4.6 1.3]

[5.6 3. 4.1 1.3]

[5.9 3.2 4.8 1.8]

[6.3 2.3 4.4 1.3]

[5.5 3.5 1.3 0.2]

[5.1 3.7 1.5 0.4]

[4.9 3.1 1.5 0.1]

[6.3 2.9 5.6 1.8]

[5.8 2.7 4.1 1. ]

[7.7 3.8 6.7 2.2]

[4.6 3.2 1.4 0.2]]

X TEST

[[5.8 2.8 5.1 2.4]

[6. 2.2 4. 1. ]

[5.5 4.2 1.4 0.2]

[7.3 2.9 6.3 1.8]

[5. 3.4 1.5 0.2]

[6.3 3.3 6. 2.5]

[5. 3.5 1.3 0.3]

[6.7 3.1 4.7 1.5]

[6.8 2.8 4.8 1.4]

[6.1 2.8 4. 1.3]

[6.1 2.6 5.6 1.4]

[6.4 3.2 4.5 1.5]

[6.1 2.8 4.7 1.2]

[6.5 2.8 4.6 1.5]

[6.1 2.9 4.7 1.4]

[4.9 3.6 1.4 0.1]

[6. 2.9 4.5 1.5]

[5.5 2.6 4.4 1.2]

[4.8 3. 1.4 0.3]

[5.4 3.9 1.3 0.4]

[5.6 2.8 4.9 2. ]

[5.6 3. 4.5 1.5]

[4.8 3.4 1.9 0.2]

[4.4 2.9 1.4 0.2]

[6.2 2.8 4.8 1.8]

[4.6 3.6 1. 0.2]

[5.1 3.8 1.9 0.4]

[6.2 2.9 4.3 1.3]

[5. 2.3 3.3 1. ]

[5. 3.4 1.6 0.4]

[6.4 3.1 5.5 1.8]

[5.4 3. 4.5 1.5]

[5.2 3.5 1.5 0.2]

[6.1 3. 4.9 1.8]

[6.4 2.8 5.6 2.2]

[5.2 2.7 3.9 1.4]

[5.7 3.8 1.7 0.3]

[6. 2.7 5.1 1.6]]

Y TRAIN

[1 1 2 0 2 0 0 1 2 2 2 2 1 2 1 1 2 2 2 2 1 2 1 0 2 1 1 1 1 2 0 0 2 1 0 0 1

0 2 1 0 1 2 1 0 2 2 2 2 0 0 2 2 0 2 0 2 2 0 0 2 0 0 0 1 2 2 0 0 0 1 1 0 0

1 0 2 1 2 1 0 2 0 2 0 0 2 0 2 1 1 1 2 2 1 1 0 1 2 2 0 1 1 1 1 0 0 0 2 1 2

0]

Y TEST

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0

1]

XNEW

[[5. 2.9 1. 0.2]]

Predicted target value: [0]

Predicted feature name: ['setosa']

XNEW

[[6. 2.2 4. 1. ]]

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 2 virginica, Predicted :[2]['virginica']

Actual : 1 versicolor, Predicted :[1]['versicolor']

Actual : 0 setosa, Predicted :[0]['setosa']

Actual : 1 versicolor, Predicted :[2]['virginica']

TEST SCORE[ACCURACY]: 0.97