**#Problem1 (b)**

import math

from plotDecBoundaries import \* # import all functions

import numpy as np

from sklearn.preprocessing import StandardScaler

import pandas as pd

np.set\_printoptions(threshold=np.inf)

# np.set\_printoptions(precision=9)

from sklearn import preprocessing

def dispose(file, choice, choice2):

xaxis = list()

yaxis = list()

label = list()

xaxistemp = list()

yaxistemp = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice2])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis, label, average

def dispose2(file, choice,choice1,choice2,choice3,choice4,choice5,choice6,choice7,choice8,choice9,choice10,choice11,choice12):

xaxis = list()

yaxis = list()

yaxis2 = list()

yaxis3 = list()

yaxis4 = list()

yaxis5 = list()

yaxis6 = list()

yaxis7 = list()

yaxis8 = list()

yaxis9 = list()

yaxis10 = list()

yaxis11 = list()

yaxis12 = list()

label = list()

xaxistemp = list()

yaxistemp = list()

yaxistemp2 = list()

yaxistemp3 = list()

yaxistemp4 = list()

yaxistemp5 = list()

yaxistemp6 = list()

yaxistemp7 = list()

yaxistemp8 = list()

yaxistemp9 = list()

yaxistemp10 = list()

yaxistemp11 = list()

yaxistemp12 = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice1])

yaxistemp2.append(line[choice2])

yaxistemp3.append(line[choice3])

yaxistemp4.append(line[choice4])

yaxistemp5.append(line[choice5])

yaxistemp6.append(line[choice6])

yaxistemp7.append(line[choice7])

yaxistemp8.append(line[choice8])

yaxistemp9.append(line[choice9])

yaxistemp10.append(line[choice10])

yaxistemp11.append(line[choice11])

yaxistemp12.append(line[choice12])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for y2 in yaxistemp2: # to float

yaxis2.append(float(y2))

for y3 in yaxistemp3: # to float

yaxis3.append(float(y3))

for y4 in yaxistemp4: # to float

yaxis4.append(float(y4))

for y5 in yaxistemp5: # to float

yaxis5.append(float(y5))

for y6 in yaxistemp6: # to float

yaxis6.append(float(y6))

for y7 in yaxistemp7: # to float

yaxis7.append(float(y7))

for y8 in yaxistemp8: # to float

yaxis8.append(float(y8))

for y9 in yaxistemp9: # to float

yaxis9.append(float(y9))

for y10 in yaxistemp10: # to float

yaxis10.append(float(y10))

for y11 in yaxistemp11: # to float

yaxis11.append(float(y11))

for y12 in yaxistemp12: # to float

yaxis12.append(float(y12))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis,yaxis2,yaxis3,yaxis4,yaxis5,yaxis6,yaxis7,yaxis8,yaxis9,yaxis10,yaxis11,yaxis12, label, average

def distance\_function(test\_val1, average\_val1, test\_val2, average\_val2):

square = (test\_val1 - average\_val1) \*\* 2 + ((test\_val2 - average\_val2) \*\* 2)

return math.sqrt(square)

def choi(tdata, tmean):

dis = [[], [], []]

outcome = [0] \* len(tdata)

for point in tdata:

dis[0].append(distance\_function(point[0], tmean[0][0], point[1], tmean[0][1]))

for point2 in tdata:

dis[1].append(distance\_function(point2[0], tmean[1][0], point2[1], tmean[1][1]))

for point3 in tdata:

dis[2].append(distance\_function(point3[0], tmean[2][0], point3[1], tmean[2][1]))

for i in range(len(tdata)):

if dis[0][i] < dis[1][i]:

if dis[1][i] < dis[2][i]:

outcome[i] = 1

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 1

else:

outcome[i] = 3

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 2

else:

if dis[1][i] < dis[2][i]:

outcome[i] = 2

else:

outcome[i] = 3

return outcome

min\_error = float("inf")

det\_k = 100

det\_j = 100

errormean = 0

errorall = list()

var = 0

temp5 = []

y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,l2,mdata2 = dispose2('wine\_train.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

training2 = []

training2.append(y0)

training2.append(y1)

training2.append(y2)

training2.append(y3)

training2.append(y4)

training2.append(y5)

training2.append(y6)

training2.append(y7)

training2.append(y8)

training2.append(y9)

training2.append(y10)

training2.append(y11)

training2.append(y12)

training2 = np.array(training2)

training2 = training2.T

mdata2 = np.array(mdata2)

l2 = np.array(l2)

print(training2)

std = StandardScaler()

std.fit(training2)

train\_std = std.fit\_transform(training2)

print('\n')

print(std.n\_samples\_seen\_)

print('\n')

print("mean:")

print(std.mean\_)

print('\n')

print("standard deviation:")

print(np.sqrt(std.var\_))

print('\n')

# print(std.scale\_)

# print('\n')

# print(train\_std)

# print(std.inverse\_transform(train\_std))

**#(d)**

import math

from plotDecBoundaries import \* # import all functions

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron

import pandas as pd

np.set\_printoptions(threshold=np.inf)

# np.set\_printoptions(precision=9)

from sklearn import preprocessing

def dispose(file, choice, choice2):

xaxis = list()

yaxis = list()

label = list()

xaxistemp = list()

yaxistemp = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice2])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis, label, average

def dispose2(file, choice,choice1,choice2,choice3,choice4,choice5,choice6,choice7,choice8,choice9,choice10,choice11,choice12):

xaxis = list()

yaxis = list()

yaxis2 = list()

yaxis3 = list()

yaxis4 = list()

yaxis5 = list()

yaxis6 = list()

yaxis7 = list()

yaxis8 = list()

yaxis9 = list()

yaxis10 = list()

yaxis11 = list()

yaxis12 = list()

label = list()

xaxistemp = list()

yaxistemp = list()

yaxistemp2 = list()

yaxistemp3 = list()

yaxistemp4 = list()

yaxistemp5 = list()

yaxistemp6 = list()

yaxistemp7 = list()

yaxistemp8 = list()

yaxistemp9 = list()

yaxistemp10 = list()

yaxistemp11 = list()

yaxistemp12 = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice1])

yaxistemp2.append(line[choice2])

yaxistemp3.append(line[choice3])

yaxistemp4.append(line[choice4])

yaxistemp5.append(line[choice5])

yaxistemp6.append(line[choice6])

yaxistemp7.append(line[choice7])

yaxistemp8.append(line[choice8])

yaxistemp9.append(line[choice9])

yaxistemp10.append(line[choice10])

yaxistemp11.append(line[choice11])

yaxistemp12.append(line[choice12])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for y2 in yaxistemp2: # to float

yaxis2.append(float(y2))

for y3 in yaxistemp3: # to float

yaxis3.append(float(y3))

for y4 in yaxistemp4: # to float

yaxis4.append(float(y4))

for y5 in yaxistemp5: # to float

yaxis5.append(float(y5))

for y6 in yaxistemp6: # to float

yaxis6.append(float(y6))

for y7 in yaxistemp7: # to float

yaxis7.append(float(y7))

for y8 in yaxistemp8: # to float

yaxis8.append(float(y8))

for y9 in yaxistemp9: # to float

yaxis9.append(float(y9))

for y10 in yaxistemp10: # to float

yaxis10.append(float(y10))

for y11 in yaxistemp11: # to float

yaxis11.append(float(y11))

for y12 in yaxistemp12: # to float

yaxis12.append(float(y12))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis,yaxis2,yaxis3,yaxis4,yaxis5,yaxis6,yaxis7,yaxis8,yaxis9,yaxis10,yaxis11,yaxis12, label, average

def distance\_function(test\_val1, average\_val1, test\_val2, average\_val2):

square = (test\_val1 - average\_val1) \*\* 2 + ((test\_val2 - average\_val2) \*\* 2)

return math.sqrt(square)

def choi(tdata, tmean):

dis = [[], [], []]

outcome = [0] \* len(tdata)

for point in tdata:

dis[0].append(distance\_function(point[0], tmean[0][0], point[1], tmean[0][1]))

for point2 in tdata:

dis[1].append(distance\_function(point2[0], tmean[1][0], point2[1], tmean[1][1]))

for point3 in tdata:

dis[2].append(distance\_function(point3[0], tmean[2][0], point3[1], tmean[2][1]))

for i in range(len(tdata)):

if dis[0][i] < dis[1][i]:

if dis[1][i] < dis[2][i]:

outcome[i] = 1

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 1

else:

outcome[i] = 3

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 2

else:

if dis[1][i] < dis[2][i]:

outcome[i] = 2

else:

outcome[i] = 3

return outcome

min\_error = float("inf")

det\_k = 100

det\_j = 100

errormean = 0

errorall = list()

var = 0

temp5 = []

for k in range(13):

for j in range(k+1, 13):

choice = k

choice2 = j

x, y, l, mdata = dispose('wine\_train.csv', choice, choice2)

with open('wine\_train.csv') as testdata:

tfile2 = testdata.readlines()

for line2 in tfile2:

temp5.append(line2.split(','))

testdata = [data[choice: choice2 + 1: choice2 - choice] for data in temp5]

tlabel = [data[13] for data in temp5]

tlabel = [int(test) for test in tlabel]

for m in range(len(testdata)):

testdata[m] = [float(data) for data in testdata[m]]

output = choi(testdata, mdata)

count = 0

for n in range(len(testdata)):

if output[n] != tlabel[n]:

count = count + 1

error = count / len(testdata)

errorall.append(error)

if error < min\_error:

min\_error = error

det\_k = choice

det\_j = choice2

for errormean2 in errorall:

errormean = errormean + errormean2

errormean = errormean / 78

for errormean3 in errorall:

var = var + ((errormean3 - errormean)\*\*2)

var = var / (78 - 1)

deviation = math.sqrt(var)

x0,x1,lx2,mdatax2 = dispose('wine\_train.csv', 0,1)

trainingx2 = []

trainingx2.append(x0)

trainingx2.append(x1)

trainingx2 = np.array(trainingx2)

trainingx2 = trainingx2.T

mdatax2 = np.array(mdatax2)

lx2 = np.array(lx2)

std2 = StandardScaler()

std2.fit(trainingx2)

train\_std2 = std2.fit\_transform(trainingx2)

print(train\_std2)

y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,l2,mdata2 = dispose2('wine\_train.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

training2 = []

training2.append(y0)

training2.append(y1)

training2.append(y2)

training2.append(y3)

training2.append(y4)

training2.append(y5)

training2.append(y6)

training2.append(y7)

training2.append(y8)

training2.append(y9)

training2.append(y10)

training2.append(y11)

training2.append(y12)

training2 = np.array(training2)

training2 = training2.T

mdata2 = np.array(mdata2)

l2 = np.array(l2)

std = StandardScaler()

std.fit(training2)

train\_std = std.fit\_transform(training2)

print(train\_std)

test\_x0,test\_x1,test\_lx2,test\_mdatax2 = dispose('wine\_test.csv', 0,1)

test\_trainingx2 = []

test\_trainingx2.append(test\_x0)

test\_trainingx2.append(test\_x1)

test\_trainingx2 = np.array(test\_trainingx2)

test\_trainingx2 = test\_trainingx2.T

test\_mdatax2 = np.array(test\_mdatax2)

test\_lx2 = np.array(test\_lx2)

test\_train\_std2 = std2.transform(test\_trainingx2)

print(test\_train\_std2)

test\_y0,test\_y1,test\_y2,test\_y3,test\_y4,test\_y5,test\_y6,test\_y7,test\_y8,test\_y9,test\_y10,test\_y11,test\_y12,test\_l2,test\_mdata2 = dispose2('wine\_test.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

test\_training2 = []

test\_training2.append(test\_y0)

test\_training2.append(test\_y1)

test\_training2.append(test\_y2)

test\_training2.append(test\_y3)

test\_training2.append(test\_y4)

test\_training2.append(test\_y5)

test\_training2.append(test\_y6)

test\_training2.append(test\_y7)

test\_training2.append(test\_y8)

test\_training2.append(test\_y9)

test\_training2.append(test\_y10)

test\_training2.append(test\_y11)

test\_training2.append(test\_y12)

test\_training2 = np.array(test\_training2)

test\_training2 = test\_training2.T

test\_mdata2 = np.array(test\_mdata2)

test\_l2 = np.array(test\_l2)

test\_train\_std = std.transform(test\_training2)

print(test\_train\_std)

# # 2 features training set

# clf = Perceptron()

# clf.fit(train\_std2, lx2)

# print('\n')

# print(" resulting 3 weight vectors:")

# weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print(weight)

# print("\n")

# acc = clf.score(train\_std2, lx2)

# print("classification accuracy:")

# print(acc)

# print('\n')

# # 2 features testing set

# clf = Perceptron()

# clf.fit(train\_std2, lx2)

# print('\n')

# print(" resulting 3 weight vectors:")

# weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print(weight)

# print("\n")

# acc = clf.score(test\_train\_std2, test\_lx2)

# print("classification accuracy:")

# print(acc)

# print('\n')

# # 13 features training set

# clf = Perceptron()

# clf.fit(train\_std, l2)

# print('\n')

# print(" resulting 3 weight vectors:")

# weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print(weight)

# print("\n")

# acc = clf.score(train\_std, l2)

# print("classification accuracy:")

# print(acc)

# print('\n')

# 13 features training set

clf = Perceptron()

clf.fit(train\_std, l2)

print('\n')

print(" resulting 3 weight vectors:")

weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

print(weight)

print("\n")

acc = clf.score(test\_train\_std, test\_l2)

print("classification accuracy:")

print(acc)

print('\n')

**#(e)**

import math

from plotDecBoundaries import \* # import all functions

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron

import pandas as pd

np.set\_printoptions(threshold=np.inf)

# np.set\_printoptions(precision=9)

from sklearn import preprocessing

def dispose(file, choice, choice2):

xaxis = list()

yaxis = list()

label = list()

xaxistemp = list()

yaxistemp = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice2])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis, label, average

def dispose2(file, choice,choice1,choice2,choice3,choice4,choice5,choice6,choice7,choice8,choice9,choice10,choice11,choice12):

xaxis = list()

yaxis = list()

yaxis2 = list()

yaxis3 = list()

yaxis4 = list()

yaxis5 = list()

yaxis6 = list()

yaxis7 = list()

yaxis8 = list()

yaxis9 = list()

yaxis10 = list()

yaxis11 = list()

yaxis12 = list()

label = list()

xaxistemp = list()

yaxistemp = list()

yaxistemp2 = list()

yaxistemp3 = list()

yaxistemp4 = list()

yaxistemp5 = list()

yaxistemp6 = list()

yaxistemp7 = list()

yaxistemp8 = list()

yaxistemp9 = list()

yaxistemp10 = list()

yaxistemp11 = list()

yaxistemp12 = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice1])

yaxistemp2.append(line[choice2])

yaxistemp3.append(line[choice3])

yaxistemp4.append(line[choice4])

yaxistemp5.append(line[choice5])

yaxistemp6.append(line[choice6])

yaxistemp7.append(line[choice7])

yaxistemp8.append(line[choice8])

yaxistemp9.append(line[choice9])

yaxistemp10.append(line[choice10])

yaxistemp11.append(line[choice11])

yaxistemp12.append(line[choice12])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for y2 in yaxistemp2: # to float

yaxis2.append(float(y2))

for y3 in yaxistemp3: # to float

yaxis3.append(float(y3))

for y4 in yaxistemp4: # to float

yaxis4.append(float(y4))

for y5 in yaxistemp5: # to float

yaxis5.append(float(y5))

for y6 in yaxistemp6: # to float

yaxis6.append(float(y6))

for y7 in yaxistemp7: # to float

yaxis7.append(float(y7))

for y8 in yaxistemp8: # to float

yaxis8.append(float(y8))

for y9 in yaxistemp9: # to float

yaxis9.append(float(y9))

for y10 in yaxistemp10: # to float

yaxis10.append(float(y10))

for y11 in yaxistemp11: # to float

yaxis11.append(float(y11))

for y12 in yaxistemp12: # to float

yaxis12.append(float(y12))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis,yaxis2,yaxis3,yaxis4,yaxis5,yaxis6,yaxis7,yaxis8,yaxis9,yaxis10,yaxis11,yaxis12, label, average

def distance\_function(test\_val1, average\_val1, test\_val2, average\_val2):

square = (test\_val1 - average\_val1) \*\* 2 + ((test\_val2 - average\_val2) \*\* 2)

return math.sqrt(square)

def choi(tdata, tmean):

dis = [[], [], []]

outcome = [0] \* len(tdata)

for point in tdata:

dis[0].append(distance\_function(point[0], tmean[0][0], point[1], tmean[0][1]))

for point2 in tdata:

dis[1].append(distance\_function(point2[0], tmean[1][0], point2[1], tmean[1][1]))

for point3 in tdata:

dis[2].append(distance\_function(point3[0], tmean[2][0], point3[1], tmean[2][1]))

for i in range(len(tdata)):

if dis[0][i] < dis[1][i]:

if dis[1][i] < dis[2][i]:

outcome[i] = 1

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 1

else:

outcome[i] = 3

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 2

else:

if dis[1][i] < dis[2][i]:

outcome[i] = 2

else:

outcome[i] = 3

return outcome

min\_error = float("inf")

det\_k = 100

det\_j = 100

errormean = 0

errorall = list()

var = 0

temp5 = []

for k in range(13):

for j in range(k+1, 13):

choice = k

choice2 = j

x, y, l, mdata = dispose('wine\_train.csv', choice, choice2)

with open('wine\_train.csv') as testdata:

tfile2 = testdata.readlines()

for line2 in tfile2:

temp5.append(line2.split(','))

testdata = [data[choice: choice2 + 1: choice2 - choice] for data in temp5]

tlabel = [data[13] for data in temp5]

tlabel = [int(test) for test in tlabel]

for m in range(len(testdata)):

testdata[m] = [float(data) for data in testdata[m]]

output = choi(testdata, mdata)

count = 0

for n in range(len(testdata)):

if output[n] != tlabel[n]:

count = count + 1

error = count / len(testdata)

errorall.append(error)

if error < min\_error:

min\_error = error

det\_k = choice

det\_j = choice2

for errormean2 in errorall:

errormean = errormean + errormean2

errormean = errormean / 78

for errormean3 in errorall:

var = var + ((errormean3 - errormean)\*\*2)

var = var / (78 - 1)

deviation = math.sqrt(var)

x0,x1,lx2,mdatax2 = dispose('wine\_train.csv', 0,1)

trainingx2 = []

trainingx2.append(x0)

trainingx2.append(x1)

trainingx2 = np.array(trainingx2)

trainingx2 = trainingx2.T

mdatax2 = np.array(mdatax2)

lx2 = np.array(lx2)

std2 = StandardScaler()

std2.fit(trainingx2)

train\_std2 = std2.fit\_transform(trainingx2)

y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,l2,mdata2 = dispose2('wine\_train.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

training2 = []

training2.append(y0)

training2.append(y1)

training2.append(y2)

training2.append(y3)

training2.append(y4)

training2.append(y5)

training2.append(y6)

training2.append(y7)

training2.append(y8)

training2.append(y9)

training2.append(y10)

training2.append(y11)

training2.append(y12)

training2 = np.array(training2)

training2 = training2.T

mdata2 = np.array(mdata2)

l2 = np.array(l2)

std = StandardScaler()

std.fit(training2)

train\_std = std.fit\_transform(training2)

test\_x0,test\_x1,test\_lx2,test\_mdatax2 = dispose('wine\_test.csv', 0,1)

test\_trainingx2 = []

test\_trainingx2.append(test\_x0)

test\_trainingx2.append(test\_x1)

test\_trainingx2 = np.array(test\_trainingx2)

test\_trainingx2 = test\_trainingx2.T

test\_mdatax2 = np.array(test\_mdatax2)

test\_lx2 = np.array(test\_lx2)

test\_train\_std2 = std2.transform(test\_trainingx2)

test\_y0,test\_y1,test\_y2,test\_y3,test\_y4,test\_y5,test\_y6,test\_y7,test\_y8,test\_y9,test\_y10,test\_y11,test\_y12,test\_l2,test\_mdata2 = dispose2('wine\_test.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

test\_training2 = []

test\_training2.append(test\_y0)

test\_training2.append(test\_y1)

test\_training2.append(test\_y2)

test\_training2.append(test\_y3)

test\_training2.append(test\_y4)

test\_training2.append(test\_y5)

test\_training2.append(test\_y6)

test\_training2.append(test\_y7)

test\_training2.append(test\_y8)

test\_training2.append(test\_y9)

test\_training2.append(test\_y10)

test\_training2.append(test\_y11)

test\_training2.append(test\_y12)

test\_training2 = np.array(test\_training2)

test\_training2 = test\_training2.T

test\_mdata2 = np.array(test\_mdata2)

test\_l2 = np.array(test\_l2)

test\_train\_std = std.transform(test\_training2)

# #2 features training set

# max\_number = 0

# max\_weight = [[], [], []]

# for i in range(100):

# clf = Perceptron()

# clf.\_allocate\_parameter\_mem(n\_classes=3, n\_features=2, coef\_init=np.random.randn(3, 2),

# intercept\_init=np.random.randn(3, ))

# clf.fit(train\_std2, lx2)

# acc = clf.score(train\_std2, lx2)

# if acc > max\_number:

# max\_number = acc

# max\_weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print('\n')

# print("classification accuracy")

# print(max\_number)

# print('\n')

# print("the final 3 weight vectors")

# print(max\_weight)

# print('\n')

# #2 features testing set

# max\_number = 0

# max\_weight = [[], [], []]

# for i in range(100):

# clf = Perceptron()

# clf.\_allocate\_parameter\_mem(n\_classes=3, n\_features=2, coef\_init=np.random.randn(3, 2),

# intercept\_init=np.random.randn(3, ))

# clf.fit(train\_std2, lx2)

# acc = clf.score(test\_train\_std2, test\_lx2)

# if acc > max\_number:

# max\_number = acc

# max\_weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print('\n')

# print("classification accuracy")

# print(max\_number)

# print('\n')

# print("the final 3 weight vectors")

# print(max\_weight)

# print('\n')

#

# #13 features training set

# max\_number = 0

# max\_weight = [[], [], []]

# for i in range(100):

# clf = Perceptron()

# clf.\_allocate\_parameter\_mem(n\_classes=3, n\_features=2, coef\_init=np.random.randn(3, 2),

# intercept\_init=np.random.randn(3, ))

# clf.fit(train\_std, l2)

# acc = clf.score(train\_std, l2)

# if acc > max\_number:

# max\_number = acc

# max\_weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

# print('\n')

# print("classification accuracy")

# print(max\_number)

# print('\n')

# print("the final 3 weight vectors")

# print(max\_weight)

# print('\n')

#13 features testing set

max\_number = 0

max\_weight = [[], [], []]

for i in range(100):

clf = Perceptron()

clf. \_allocate\_parameter\_mem(n\_classes = 3, n\_features = 2, coef\_init = np.random. randn(3,2),intercept\_init = np.random.randn(3,))

clf.fit(train\_std, l2)

acc = clf.score(test\_train\_std, test\_l2)

if acc > max\_number:

max\_number = acc

max\_weight = np.concatenate((clf.intercept\_.reshape(3, 1), clf.coef\_), axis=1)

print('\n')

print("classification accuracy")

print(max\_number)

print('\n')

print("the final 3 weight vectors")

print(max\_weight)

print('\n')

**#(g)(h)**

import math

from plotDecBoundaries import \* # import all functions

import numpy as np

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression

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

import pandas as pd

from sklearn.multiclass import OneVsRestClassifier

np.set\_printoptions(threshold=np.inf)

# np.set\_printoptions(precision=9)

from sklearn import preprocessing

def dispose(file, choice, choice2):

xaxis = list()

yaxis = list()

label = list()

xaxistemp = list()

yaxistemp = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice2])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis, label, average

def dispose2(file, choice,choice1,choice2,choice3,choice4,choice5,choice6,choice7,choice8,choice9,choice10,choice11,choice12):

xaxis = list()

yaxis = list()

yaxis2 = list()

yaxis3 = list()

yaxis4 = list()

yaxis5 = list()

yaxis6 = list()

yaxis7 = list()

yaxis8 = list()

yaxis9 = list()

yaxis10 = list()

yaxis11 = list()

yaxis12 = list()

label = list()

xaxistemp = list()

yaxistemp = list()

yaxistemp2 = list()

yaxistemp3 = list()

yaxistemp4 = list()

yaxistemp5 = list()

yaxistemp6 = list()

yaxistemp7 = list()

yaxistemp8 = list()

yaxistemp9 = list()

yaxistemp10 = list()

yaxistemp11 = list()

yaxistemp12 = list()

labeltemp = list()

average = list()

temp = list()

temp2 = list()

temp3 = list()

with open(file) as train:

tfile = train.read().splitlines()

for line in tfile:

line = line.split(',') # cut to piece

xaxistemp.append(line[choice])

yaxistemp.append(line[choice1])

yaxistemp2.append(line[choice2])

yaxistemp3.append(line[choice3])

yaxistemp4.append(line[choice4])

yaxistemp5.append(line[choice5])

yaxistemp6.append(line[choice6])

yaxistemp7.append(line[choice7])

yaxistemp8.append(line[choice8])

yaxistemp9.append(line[choice9])

yaxistemp10.append(line[choice10])

yaxistemp11.append(line[choice11])

yaxistemp12.append(line[choice12])

labeltemp.append(line[13])

for x in xaxistemp: # to float

xaxis.append(float(x))

for y in yaxistemp: # to float

yaxis.append(float(y))

for y2 in yaxistemp2: # to float

yaxis2.append(float(y2))

for y3 in yaxistemp3: # to float

yaxis3.append(float(y3))

for y4 in yaxistemp4: # to float

yaxis4.append(float(y4))

for y5 in yaxistemp5: # to float

yaxis5.append(float(y5))

for y6 in yaxistemp6: # to float

yaxis6.append(float(y6))

for y7 in yaxistemp7: # to float

yaxis7.append(float(y7))

for y8 in yaxistemp8: # to float

yaxis8.append(float(y8))

for y9 in yaxistemp9: # to float

yaxis9.append(float(y9))

for y10 in yaxistemp10: # to float

yaxis10.append(float(y10))

for y11 in yaxistemp11: # to float

yaxis11.append(float(y11))

for y12 in yaxistemp12: # to float

yaxis12.append(float(y12))

for l in labeltemp: # to float

label.append(int(l))

for j in range(len(label)):

if label[j] == 1:

temp.append(j)

if label[j] == 2:

temp2.append(j)

if label[j] == 3:

temp3.append(j)

xaverage = sum(xaxis[min(temp): max(temp) + 1]) / len(temp)

yaverage = sum(yaxis[min(temp): max(temp) + 1]) / len(temp)

xaverage2 = sum(xaxis[min(temp2): max(temp2) + 1]) / len(temp2)

yaverage2 = sum(yaxis[min(temp2): max(temp2) + 1]) / len(temp2)

xaverage3 = sum(xaxis[min(temp3): max(temp3) + 1]) / len(temp3)

yaverage3 = sum(yaxis[min(temp3): max(temp3) + 1]) / len(temp3)

average = [[xaverage, yaverage], [xaverage2, yaverage2], [xaverage3, yaverage3]]

return xaxis, yaxis,yaxis2,yaxis3,yaxis4,yaxis5,yaxis6,yaxis7,yaxis8,yaxis9,yaxis10,yaxis11,yaxis12, label, average

def distance\_function(test\_val1, average\_val1, test\_val2, average\_val2):

square = (test\_val1 - average\_val1) \*\* 2 + ((test\_val2 - average\_val2) \*\* 2)

return math.sqrt(square)

def choi(tdata, tmean):

dis = [[], [], []]

outcome = [0] \* len(tdata)

for point in tdata:

dis[0].append(distance\_function(point[0], tmean[0][0], point[1], tmean[0][1]))

for point2 in tdata:

dis[1].append(distance\_function(point2[0], tmean[1][0], point2[1], tmean[1][1]))

for point3 in tdata:

dis[2].append(distance\_function(point3[0], tmean[2][0], point3[1], tmean[2][1]))

for i in range(len(tdata)):

if dis[0][i] < dis[1][i]:

if dis[1][i] < dis[2][i]:

outcome[i] = 1

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 1

else:

outcome[i] = 3

else:

if dis[0][i] < dis[2][i]:

outcome[i] = 2

else:

if dis[1][i] < dis[2][i]:

outcome[i] = 2

else:

outcome[i] = 3

return outcome

min\_error = float("inf")

det\_k = 100

det\_j = 100

errormean = 0

errorall = list()

var = 0

temp5 = []

for k in range(13):

for j in range(k+1, 13):

choice = k

choice2 = j

x, y, l, mdata = dispose('wine\_train.csv', choice, choice2)

with open('wine\_train.csv') as testdata:

tfile2 = testdata.readlines()

for line2 in tfile2:

temp5.append(line2.split(','))

testdata = [data[choice: choice2 + 1: choice2 - choice] for data in temp5]

tlabel = [data[13] for data in temp5]

tlabel = [int(test) for test in tlabel]

for m in range(len(testdata)):

testdata[m] = [float(data) for data in testdata[m]]

output = choi(testdata, mdata)

count = 0

for n in range(len(testdata)):

if output[n] != tlabel[n]:

count = count + 1

error = count / len(testdata)

errorall.append(error)

if error < min\_error:

min\_error = error

det\_k = choice

det\_j = choice2

for errormean2 in errorall:

errormean = errormean + errormean2

errormean = errormean / 78

for errormean3 in errorall:

var = var + ((errormean3 - errormean)\*\*2)

var = var / (78 - 1)

deviation = math.sqrt(var)

x0,x1,lx2,mdatax2 = dispose('wine\_train.csv', 0,1)

trainingx2 = []

trainingx2.append(x0)

trainingx2.append(x1)

trainingx2 = np.array(trainingx2)

trainingx2 = trainingx2.T

mdatax2 = np.array(mdatax2)

lx2 = np.array(lx2)

std2 = StandardScaler()

std2.fit(trainingx2)

train\_std2 = std2.fit\_transform(trainingx2)

y0,y1,y2,y3,y4,y5,y6,y7,y8,y9,y10,y11,y12,l2,mdata2 = dispose2('wine\_train.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

training2 = []

training2.append(y0)

training2.append(y1)

training2.append(y2)

training2.append(y3)

training2.append(y4)

training2.append(y5)

training2.append(y6)

training2.append(y7)

training2.append(y8)

training2.append(y9)

training2.append(y10)

training2.append(y11)

training2.append(y12)

training2 = np.array(training2)

training2 = training2.T

mdata2 = np.array(mdata2)

l2 = np.array(l2)

std = StandardScaler()

std.fit(training2)

train\_std = std.fit\_transform(training2)

test\_x0,test\_x1,test\_lx2,test\_mdatax2 = dispose('wine\_test.csv', 0,1)

test\_trainingx2 = []

test\_trainingx2.append(test\_x0)

test\_trainingx2.append(test\_x1)

test\_trainingx2 = np.array(test\_trainingx2)

test\_trainingx2 = test\_trainingx2.T

test\_mdatax2 = np.array(test\_mdatax2)

test\_lx2 = np.array(test\_lx2)

test\_train\_std2 = std2.transform(test\_trainingx2)

test\_y0,test\_y1,test\_y2,test\_y3,test\_y4,test\_y5,test\_y6,test\_y7,test\_y8,test\_y9,test\_y10,test\_y11,test\_y12,test\_l2,test\_mdata2 = dispose2('wine\_test.csv', 0,1,2,3,4,5,6,7,8,9,10,11,12)

test\_training2 = []

test\_training2.append(test\_y0)

test\_training2.append(test\_y1)

test\_training2.append(test\_y2)

test\_training2.append(test\_y3)

test\_training2.append(test\_y4)

test\_training2.append(test\_y5)

test\_training2.append(test\_y6)

test\_training2.append(test\_y7)

test\_training2.append(test\_y8)

test\_training2.append(test\_y9)

test\_training2.append(test\_y10)

test\_training2.append(test\_y11)

test\_training2.append(test\_y12)

test\_training2 = np.array(test\_training2)

test\_training2 = test\_training2.T

test\_mdata2 = np.array(test\_mdata2)

test\_l2 = np.array(test\_l2)

test\_train\_std = std.transform(test\_training2)

# # unnormalized data 2features test data

# class MSE\_binary(LinearRegression):

# def \_\_init\_\_(self):

# print("Calling newly created MSE binary function")

# super(MSE\_binary, self).\_\_init\_\_()

# def predict(self,X):

# thr = 0.5

# y = self.\_decision\_function(X)

# for i in range((len(y))):

# if y[i] <= thr:

# y[i] = 0

# elif y[i] >thr:

# y[i] = 1

# return y

#

# binary\_model = MSE\_binary()

# model = OneVsRestClassifier(binary\_model)

# model.fit(trainingx2, lx2)

# macc = model.score(test\_trainingx2, test\_lx2)

# print('\n')

# print(macc)

# # unnormalized data 13features test data

# class MSE\_binary(LinearRegression):

# def \_\_init\_\_(self):

# print("Calling newly created MSE binary function")

# super(MSE\_binary, self).\_\_init\_\_()

# def predict(self,X):

# thr = 0.5

# y = self.\_decision\_function(X)

# for i in range((len(y))):

# if y[i] <= thr:

# y[i] = 0

# elif y[i] >thr:

# y[i] = 1

# return y

#

# binary\_model = MSE\_binary()

# model = OneVsRestClassifier(binary\_model)

# model.fit(training2, l2)

# macc = model.score(test\_training2, test\_l2)

# print('\n')

# print(macc)

# # standardized data 2features test data

# class MSE\_binary(LinearRegression):

# def \_\_init\_\_(self):

# super(MSE\_binary, self).\_\_init\_\_()

# print("Calling newly created MSE binary function")

# def predict(self,X):

# thr = 0.5

# y = self.\_decision\_function(X)

# for i in range((len(y))):

# if y[i] <= thr:

# y[i] = 0

# elif y[i] >thr:

# y[i] = 1

# return y

#

# binary\_model = MSE\_binary()

# model = OneVsRestClassifier(binary\_model)

# model.fit(train\_std2, lx2)

# macc = model.score(test\_train\_std2, test\_lx2)

# print('\n')

# print(macc)

# standardized data 13features test data

class MSE\_binary(LinearRegression):

def \_\_init\_\_(self):

super(MSE\_binary, self).\_\_init\_\_()

print("Calling newly created MSE binary function")

def predict(self,X):

thr = 0.5

y = self.\_decision\_function(X)

for i in range((len(y))):

if y[i] <= thr:

y[i] = 0

elif y[i] >thr:

y[i] = 1

return y

binary\_model = MSE\_binary()

model = OneVsRestClassifier(binary\_model)

model.fit(train\_std, l2)

macc = model.score(test\_train\_std, test\_l2)

print('\n')

print(macc)