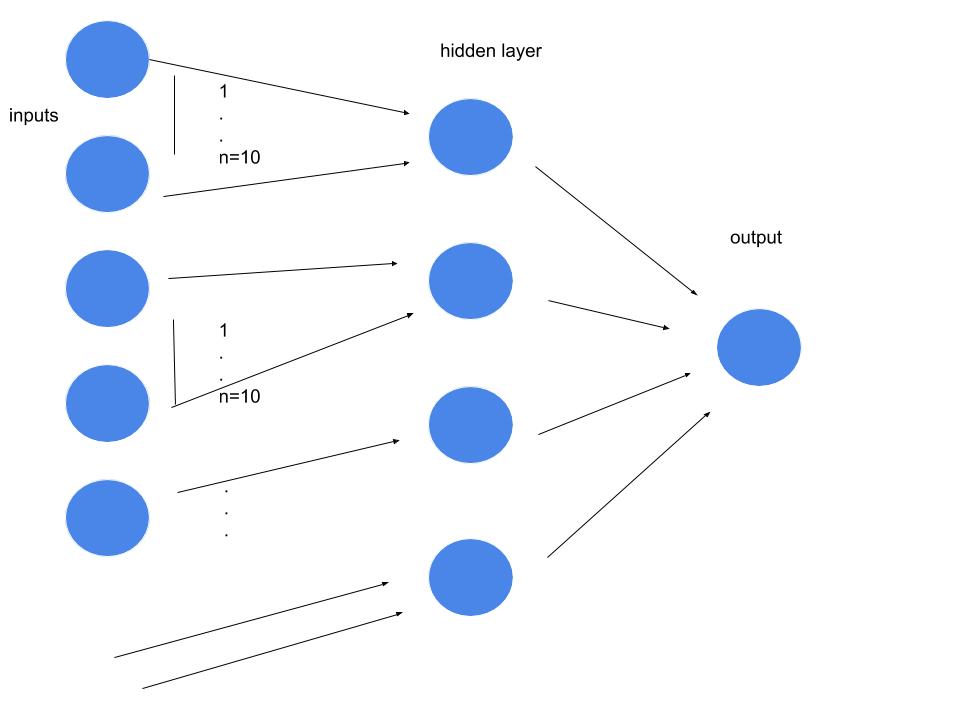
**מגישים:** שי קאיקוב-312531031.מתן צברי-302548565.יוני כהן-204507743.

**Report-Backpropagation-question 3:**

In this part I separate the results according to the number of the iterations of the loop.

The architecture of the network was-in first 4 until 36 data(row) that enter as inputs to the network(data between 1 and 3 I didn’t enter in the reason that they are the id ,age and output).the inputs go to the first hidden layer .the hidden layer consist four neurons. the four neurons are pass activation(sigmoid function) and than go to the last layer-the output layer .the reason that I decided on this architecture because my intuition told me that the first neuron in the first hidden layer will belong to the 4-13 data in the reason that they averages and this neuron will be work well with that. and in this way I took the next 10 data and put it in the second neuron -in the hidden layer. the third 10 data go to the that third neuron. and the last neuron was belong to the last two data(because they were not attached to the rest of the data so I gather them into inputs to the fourth neuron in the first hidden layer).the four neurons are the inputs(after I activate the activation function on them) for the last layer(the output neuron) .here the picture that illustrate my architecture:

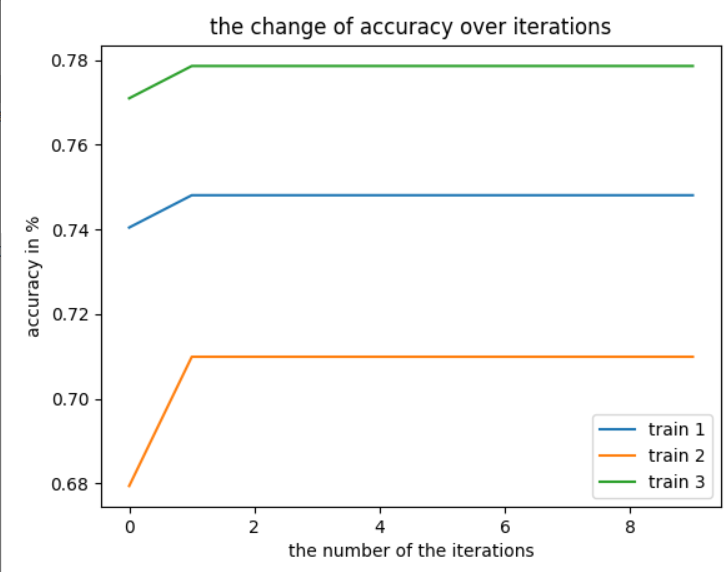
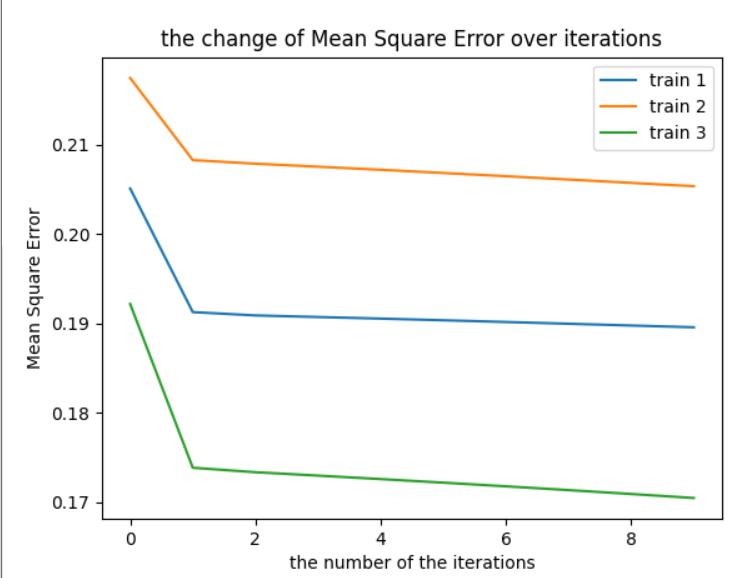


The learning rate was 0.1(alpha). actually the learning rate had little effect on my network but eventually I pick 0.1 because it still was a little impact. the value-0.1 was the best learning rate for my network(a=0.1).

The activation function was sigmoid-I picked that in the reason that I know well this function.

The bias input was 1.I initialized all the weight to zero and where was ? in the data I change to 0 in order that will not impact my results(because we don’t know the input!).in train 1 I chose the first 66% of the data and in the test the last 33% of the data.in train 2 I chose the 66% of the data in the middle and in the test I took the 33% of the data in the start and in the end.in train 3 I chose the last 66% of the data and in the test I took the first 33% of the data. In the balance way I chose(in the reason that the positive data were few) to pick according to the order because they are proportion on the train and in the test(according to the order)

**Iterations=10:**

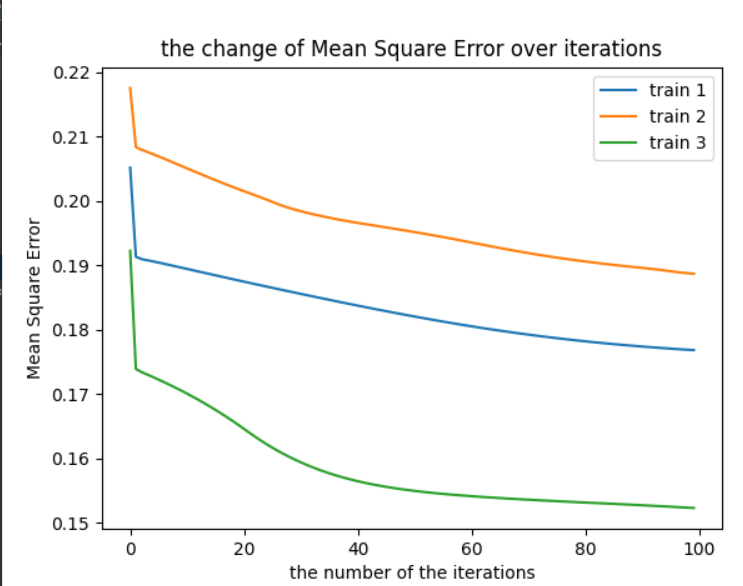


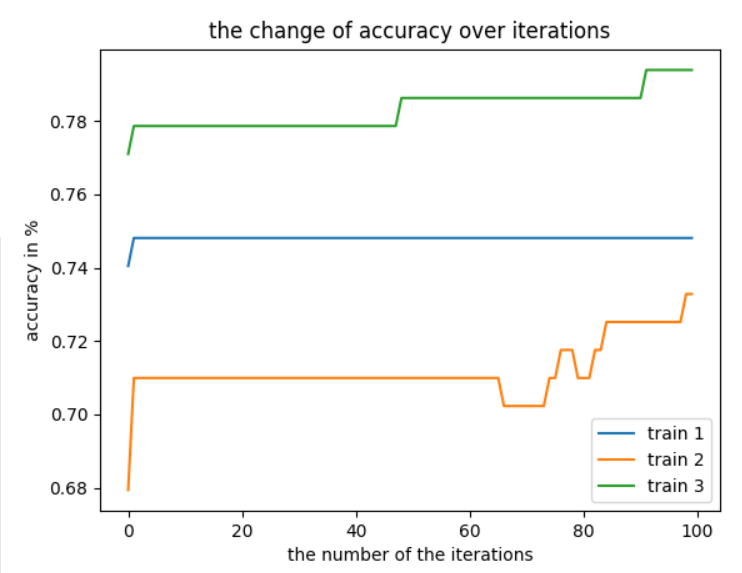
the accuracy of test 1 is(in %): 0.7878787878787878 Mean Square Error of test 1 is: 0.16639934301260617 the accuracy of test 2 is(in %): 0.8636363636363636 Mean Square Error of test 2 is: 0.14082806533422018 the accuracy of test 3 is(in %): 0.727272727272727 Mean Square Error of test 3 is: 0.2011412950407343 the average accuracy(3 tests):0.7928 the std accuracy(3 tests):0.055800

|  |  |  |
| --- | --- | --- |
| Predict: no | Predict: yes | **The 3 tests** |
| 0.207(ave)/0.055(std) | 0(ave)/0(std) | Actual: yes |
| 0.792(ave)/0.055(std) | 0(ave)/0(std) | Actual: no |

the program run in seconds: 2.93054223060607

**Iterations=100:**



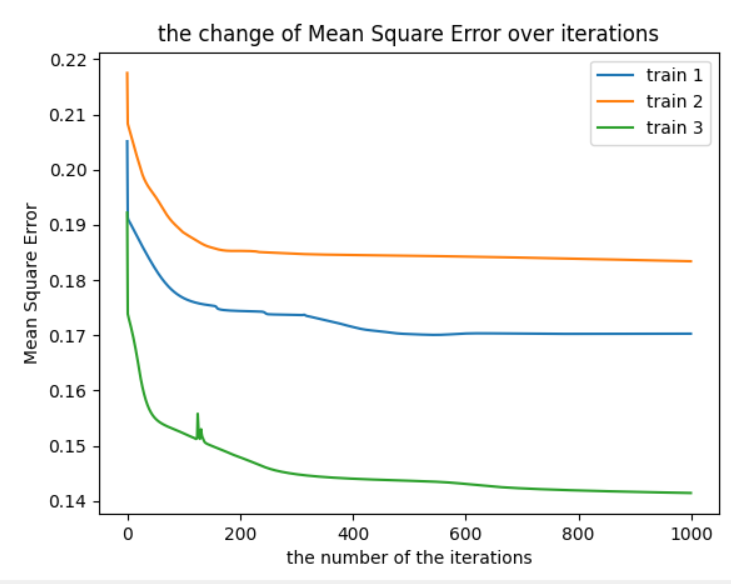


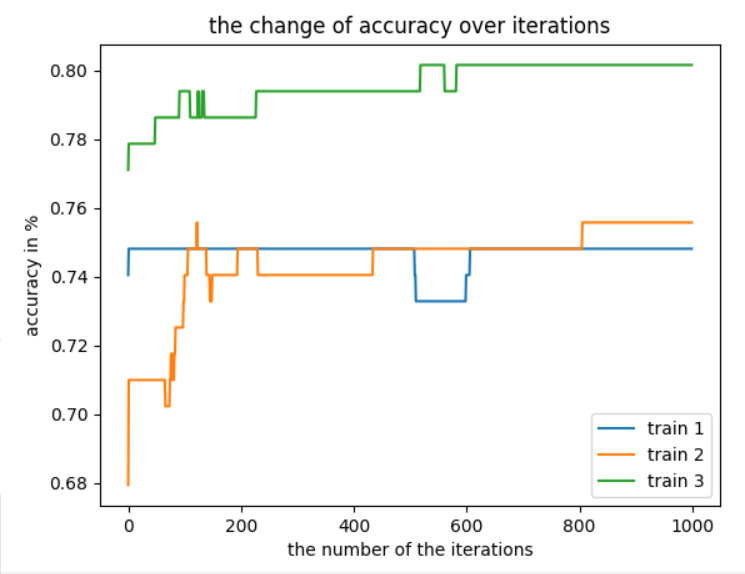
the accuracy of test 1 is(in %): 0.7878787878787878 Mean Square Error of test 1 is: 0.1527159468259893 the accuracy of test 2 is(in %): 0.8636363636363636 Mean Square Error of test 2 is: 0.12514734999022684 the accuracy of test 3 is(in %): 0.7272727272727273 Mean Square Error of test 3 is: 0.21215717795849845 the average accuracy(3 tests):0.7929 the std accuracy(3 tests):0.0557

|  |  |  |
| --- | --- | --- |
| Predict: no | Predict: yes | **The 3 tests** |
| 0.207(ave)/0.055(std) | 0(ave)/0(std) | Actual: yes |
| 0.792(ave)/0.055(std) | 0(ave)/0(std) | Actual: no |

the program run in seconds: 3.709667921066284

**Iterations=1000:**





the accuracy of test 1 is(in %): 0.7878787878787878 Mean Square Error of test 1 is: 0.14818854826094052 the accuracy of test 2 is(in %): 0.8181818181818182 Mean Square Error of test 2 is: 0.12328625633978098 the accuracy of test 3 is(in %): 0.7121212121212121 Mean Square Error of test 3 is: 0.22950182293039303 the average accuracy(3 tests): 0.77267333333 the std accuracy(3 tests): 0.044577747313604

|  |  |  |
| --- | --- | --- |
| Predict: no | Predict: yes | **The 3 tests** |
| 0.19696(ave)/0.0566(std) | 0.01010(ave)/0.00714(std) | Actual: yes |
| 0.7626(ave)/0.0468(std) | 0.0303(ave)/0.0247(std) | Actual: no |

the program run in seconds: 29.63913106918335

between the 101 iterations until 999 I don’t treat to in the reason that their result are similar with the iterations=10(but with little bit difference).

**Conclusions:**

As we can see in iteration=10 -in the start was very strong improvement. After that- there was no almost improvement.

In iteration=100 in the start was improvement and in the end and the result were the same like in iteration=10.

In iteration=1000 we can see something interesting, despite the results were little bit disappointed, from the table-we can see that predict-yes and actual yes and no- have been treated- maybe that because the lot of iterations that the algorithm succeed little bit to learn about the positive data(that were very few).in the others iteration we see that the results were 0- I assume that the algorithm didn't succeed in short time to learn a few data(positive data).

**Code(python):**

import math  
import numpy as np  
import pandas as pd  
import random  
import matplotlib.pyplot as plt  
import time  
from matplotlib.colors import ListedColormap  
  
#a=alpha  
#Sampels\_X-inputs matrix  
#Sampels\_y-outputs array  
#num\_iter-number of iterations  
  
  
  
indication=0  
  
def update(a,output,inputs,lis,W1\_FirstTen\_firstLyer,  
 W2\_SecondTen\_firstLyer,W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1):  
 Lk=(output-lis[1][4])\*Derative\_f(lis[0][4])  
  
  
 L\_in\_2=[0,0,0,0]  
 L2=[0,0,0,0]  
 DelteWend=np.zeros(len(Wend1))  
 for i in range(0,len(DelteWend)-1):  
 DelteWend[i]=(a\*Lk\*lis[1][i])  
 L\_in\_2[i]=(Lk\*Wend1[i])  
 L2[i]=(L\_in\_2[i])\*Derative\_f(lis[0][i])  
 DelteWend[len(DelteWend)-1]=(a\*Lk)  
  
  
  
  
 deltaW1=np.zeros(len(W1\_FirstTen\_firstLyer))  
 for i in range (0,len(deltaW1)-1):  
 deltaW1[i]=a\*L2[0]\*inputs[i]  
 deltaW1[len(deltaW1)-1]=a\*L2[0]  
  
 deltaW2 = np.zeros(len(W2\_SecondTen\_firstLyer))  
 for i in range(0, len(deltaW2) - 1):  
 deltaW2[i] = a \* L2[1] \* inputs[10+i]  
 deltaW2[len(deltaW2) - 1] = a \* L2[1]  
  
 deltaW3 = np.zeros(len(W3\_ThirdTen\_firstLyer))  
 for i in range(0, len(deltaW3) - 1):  
 deltaW3[i] = a \* L2[2] \* inputs[20 + i]  
 deltaW3[len(deltaW3) - 1] = a \* L2[2]  
  
 deltaW4 = np.zeros(len(W4\_FourthTen\_firstLyer))  
 for i in range(0, len(deltaW4) - 1):  
 deltaW4[i] = a \* L2[3] \* inputs[30 + i]  
 deltaW4[len(deltaW4) - 1] = a \* L2[3]  
  
 Wend1=Wend1+DelteWend  
 W4\_FourthTen\_firstLyer=W4\_FourthTen\_firstLyer+deltaW4  
 W3\_ThirdTen\_firstLyer=W3\_ThirdTen\_firstLyer+deltaW3  
 W2\_SecondTen\_firstLyer=W2\_SecondTen\_firstLyer+deltaW2  
 W1\_FirstTen\_firstLyer=W1\_FirstTen\_firstLyer+deltaW1  
 return [W1\_FirstTen\_firstLyer,W2\_SecondTen\_firstLyer,W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1]  
  
def Derative\_f(x):  
 m=(f(x)\*(1-f(x)))  
 return m  
  
def f(x):  
  
 y= (1/(1+(math.exp(-x))))  
 return y  
  
  
def Feedforward(inputsArr,W1\_FirstTen\_firstLyer,W2\_SecondTen\_firstLyer,  
 W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1):  
 N1\_hiddenLayer=0  
  
 for i in range(0,len(W1\_FirstTen\_firstLyer)-1):  
 N1\_hiddenLayer=N1\_hiddenLayer+(inputsArr[i]\*W1\_FirstTen\_firstLyer[i])  
 N1\_hiddenLayer=N1\_hiddenLayer+(W1\_FirstTen\_firstLyer[len(W1\_FirstTen\_firstLyer)-1]\*1)  
 Z1\_hiddenLayer = f(N1\_hiddenLayer)  
  
 N2\_hiddenLayer = 0  
 for i in range(0, len(W2\_SecondTen\_firstLyer) - 1):  
 N2\_hiddenLayer = N2\_hiddenLayer + (inputsArr[10+i] \* W2\_SecondTen\_firstLyer[i])  
 N2\_hiddenLayer = N2\_hiddenLayer + (W2\_SecondTen\_firstLyer[len(W2\_SecondTen\_firstLyer) - 1] \* 1)  
 Z2\_hiddenLayer = f(N2\_hiddenLayer)  
  
 N3\_hiddenLayer = 0  
 for i in range(0, len(W3\_ThirdTen\_firstLyer) - 1):  
 N3\_hiddenLayer = N3\_hiddenLayer + (inputsArr[20+i] \* W3\_ThirdTen\_firstLyer[i])  
 N3\_hiddenLayer = N3\_hiddenLayer + (W3\_ThirdTen\_firstLyer[len(W3\_ThirdTen\_firstLyer) - 1] \* 1)  
 Z3\_hiddenLayer = f(N3\_hiddenLayer)  
  
  
 N4\_hiddenLayer = 0  
 for i in range(0, len(W4\_FourthTen\_firstLyer) - 1):  
 N4\_hiddenLayer = N4\_hiddenLayer + (inputsArr[30 + i] \* W4\_FourthTen\_firstLyer[i])  
 N4\_hiddenLayer = N4\_hiddenLayer + (W4\_FourthTen\_firstLyer[len(W4\_FourthTen\_firstLyer) - 1] \* 1)  
 Z4\_hiddenLayer = f(N4\_hiddenLayer)  
  
 outputp=0  
 outputp = (Z1\_hiddenLayer\*Wend1[0])+(Z2\_hiddenLayer\*Wend1[1])+(Z3\_hiddenLayer\*Wend1[2])+(Z4\_hiddenLayer\*Wend1[3])  
 outputp=outputp+(Wend1[4]\*1)  
 Aoutput=f(outputp)  
  
 return [[N1\_hiddenLayer,N2\_hiddenLayer,N3\_hiddenLayer,N4\_hiddenLayer,outputp],  
 [Z1\_hiddenLayer,Z2\_hiddenLayer,Z3\_hiddenLayer,Z4\_hiddenLayer,  
 Aoutput]]  
  
  
  
  
def fit(num\_iter,a,Sampels\_X,Sampels\_Y):  
 W1\_FirstTen\_firstLyer=np.zeros(11);  
 W2\_SecondTen\_firstLyer=np.zeros(11);  
 W3\_ThirdTen\_firstLyer=np.zeros(11);  
 W4\_FourthTen\_firstLyer=np.zeros(3);  
 Wend1=np.zeros(5);  
 listACC=[]  
 listERSQ=[]  
 listITER=[]  
 for it in range(0,num\_iter):  
 AERQ=0  
 ACC=0  
 for i in range(0,len(Sampels\_X)):  
 lis=Feedforward(Sampels\_X[i],W1\_FirstTen\_firstLyer,  
 W2\_SecondTen\_firstLyer,W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1)  
 lis1=update(a,Sampels\_Y[i],Sampels\_X[i],lis,W1\_FirstTen\_firstLyer,  
 W2\_SecondTen\_firstLyer,W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1)  
 W1\_FirstTen\_firstLyer=lis1[0]  
 W2\_SecondTen\_firstLyer=lis1[1]  
 W3\_ThirdTen\_firstLyer=lis1[2]  
 W4\_FourthTen\_firstLyer=lis1[3]  
 Wend1=lis1[4]  
 AERQ=AERQ+((Sampels\_Y[i]-lis[1][4])\*(Sampels\_Y[i]-lis[1][4]))  
  
 if lis[1][4]>=0.5:  
 A=1  
 if A==Sampels\_Y[i]:  
 ACC=ACC+1  
 elif lis[1][4]<0.5:  
 A=0  
 if A==Sampels\_Y[i]:  
 ACC=ACC+1  
  
 ACC=(ACC/len(Sampels\_Y))  
 AERQ=(AERQ/len(Sampels\_Y))  
 if indication==0:  
 listACC.append(ACC)  
 listERSQ.append(AERQ)  
 listITER.append(it)  
 elif indication==1:  
 listACC.append(ACC)  
 listERSQ.append(AERQ)  
 listITER.append(it)  
 elif indication==2:  
 listACC.append(ACC)  
 listERSQ.append(AERQ)  
 listITER.append(it)  
  
  
 averges = [listACC, listERSQ, listITER]  
  
 weg=[W1\_FirstTen\_firstLyer,W2\_SecondTen\_firstLyer,W3\_ThirdTen\_firstLyer,W4\_FourthTen\_firstLyer,Wend1]  
  
 return averges,weg  
  
  
def test(lis1,Sampels\_X,Sampels\_Y):  
 sumT=0  
 AERQ=0  
 pa1=0  
 pa2=0  
 pa3=0  
 pa4=0  
 for i in range(0,len(Sampels\_X)):  
 lis = Feedforward(Sampels\_X[i], lis1[0],lis1[1],lis1[2],lis1[3],lis1[4])  
  
 if lis[1][4]>=0.5:  
 py=1  
 if py==Sampels\_Y[i]:  
 sumT=sumT+1  
 pa1=pa1+1  
 else:  
 pa3=pa3+1  
 else:  
 py=0  
 if py==Sampels\_Y[i]:  
 sumT=sumT+1  
 pa4=pa4+1  
 else:  
 pa2=pa2+1  
 AERQ = AERQ + ((Sampels\_Y[i] - lis[1][4]) \* (Sampels\_Y[i] - lis[1][4]))  
 pa1=(pa1/len(Sampels\_Y))  
 pa2 = (pa2 / len(Sampels\_Y))  
 pa3 = (pa3 / len(Sampels\_Y))  
 pa4 = (pa4 / len(Sampels\_Y))  
 a= (sumT/(len(Sampels\_Y)))  
 b=(AERQ/len(Sampels\_Y))  
 c=[pa1,pa2,pa3,pa4]  
 return a,b,c  
  
  
  
start\_time=time.time()  
file=pd.read\_csv('https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wpbc.data')  
len\_rows\_train=(int)(0.666666\*len(file))  
line\_y=file.iloc[0:len(file),[1]]  
  
  
line\_y=np.where(line\_y=='N',0,1)  
temp=[]  
  
for i in range(len(line\_y)):  
 temp.append(line\_y[i][0])  
  
  
  
line\_y=np.array(temp)  
  
  
  
  
  
  
  
  
  
ave1=[]  
ave2=[]  
ave3=[]  
  
time\_of\_train\_test=0  
while(time\_of\_train\_test<3):  
  
 a=0.1  
 iter=100  
 if time\_of\_train\_test==0:  
 x = np.array(file.iloc[0:len\_rows\_train, 3:(len(file.columns))])  
 y = np.zeros(len\_rows\_train)  
 for i in range(0, len\_rows\_train):  
 y[i] = line\_y[i]  
  
 for i in range(len(x)):  
 for j in range(len(x[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x[i][j] = float(x[i][j]) / 1500  
 if x[i][j] == '?':  
 x[i][j] = 0  
 x[i][j] = float(x[i][j])  
  
 ave1,lis1 = fit(iter, a, x, y)  
  
 x = np.array(file.iloc[len\_rows\_train:len(file), 3:(len(file.columns))])  
 y = np.zeros(len(file)-len\_rows\_train)  
 for i in range(0,len(file)-len\_rows\_train):  
 y[i] = line\_y[i+len\_rows\_train]  
  
 for i in range(len(x)):  
 for j in range(len(x[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x[i][j] = float(x[i][j]) / 1500  
 if x[i][j] == '?':  
 x[i][j] = 0  
 x[i][j] = float(x[i][j])  
 acc, erq,ta1 = test(lis1, x, y)  
 print("the accuracy of test 1 is(in %): ", acc)  
 print("Mean Square Error of test 1 is: ", erq)  
 finalav1 = acc  
  
 elif time\_of\_train\_test==1:  
  
 x = np.array(file.iloc[int((len(file) - len\_rows\_train)/2):len\_rows\_train+(int((len(file) - len\_rows\_train)/2)),  
 3:(len(file.columns))])  
 y = np.zeros(len\_rows\_train)  
 for i in range(0, len\_rows\_train):  
 y[i] = line\_y[i+int((len(file) - len\_rows\_train)/2)]  
  
 for i in range(len(x)):  
 for j in range(len(x[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x[i][j] = float(x[i][j]) / 1500  
 if x[i][j] == '?':  
 x[i][j] = 0  
 x[i][j] = float(x[i][j])  
  
  
 ave2,lis1 = fit(iter, a, x, y)  
  
 x1 = np.array(file.iloc[0:int((len(file) - len\_rows\_train)/2), 3:(len(file.columns))])  
 y1 = np.zeros(int((len(file) - len\_rows\_train)/2))  
 for i in range(0, len(y1)):  
 y1[i] = line\_y[i]  
  
 for i in range(len(x1)):  
 for j in range(len(x1[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x1[i][j] = float(x1[i][j]) / 1500  
 if x1[i][j] == '?':  
 x1[i][j] = 0  
 x1[i][j] = float(x1[i][j])  
  
 x2 = np.array(file.iloc[len\_rows\_train+int((len(file) - len\_rows\_train) / 2):len(file),  
 3:(len(file.columns))])  
 y2 = np.zeros(int((len(file) - len\_rows\_train) / 2))  
 for i in range(0, len(y2)):  
 y2[i] = line\_y[i+len\_rows\_train+int((len(file) - len\_rows\_train) / 2)]  
  
 for i in range(len(x2)):  
 for j in range(len(x2[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x2[i][j] = float(x2[i][j]) / 1500  
 if x2[i][j] == '?':  
 x2[i][j] = 0  
 x2[i][j] = float(x2[i][j])  
  
  
 x=np.zeros([len(file)-len\_rows\_train,len(x1[1])])  
 y=np.zeros(len(file)-len\_rows\_train)  
 for i in range(len(x1)):  
 for j in range(len(x1[i])):  
 x[i][j]=x1[i][j]  
 y[i]=y1[i]  
  
 for i in range(len(x2)):  
 for j in range(len(x2[i])):  
 x[i+len(x1)][j] = x2[i][j]  
 y[i+len(y1)] = y2[i]  
  
  
  
  
 acc,erq,ta2 = test(lis1, x, y)  
 print("the accuracy of test 2 is(in %): ",acc)  
 print("Mean Square Error of test 2 is: ",erq)  
 finalav2 = acc  
  
  
  
 elif time\_of\_train\_test==2:  
  
 x = np.array(file.iloc[len(file)-len\_rows\_train:len(file), 3:(len(file.columns))])  
 y = np.zeros(len\_rows\_train)  
 for i in range(0, len\_rows\_train):  
 y[i] = line\_y[i+len(file)-len\_rows\_train]  
  
 for i in range(len(x)):  
 for j in range(len(x[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x[i][j] = float(x[i][j]) / 1500  
 if x[i][j] == '?':  
 x[i][j] = 0  
 x[i][j] = float(x[i][j])  
  
  
 ave3,lis1 = fit(iter, a, x, y)  
  
 x = np.array(file.iloc[0:len(file)-len\_rows\_train, 3:(len(file.columns))])  
 y = np.zeros(len(file) - len\_rows\_train)  
 for i in range(0, len(file) - len\_rows\_train):  
 y[i] = line\_y[i]  
  
 for i in range(len(x)):  
 for j in range(len(x[i])):  
 if j == 2 or j == 3 or j == 22 or j == 23:  
 x[i][j] = float(x[i][j]) / 1500  
 if x[i][j] == '?':  
 x[i][j] = 0  
 x[i][j] = float(x[i][j])  
 acc,erq,ta3 = test(lis1, x, y)  
 finalav3 = acc  
 print("the accuracy of test 3 is(in %): ", acc)  
 print("Mean Square Error of test 3 is: ", erq)  
 print("the average accuracy(3 tests): ", ((finalav3 + finalav2 + finalav1) / 3))  
 arr = [finalav1, finalav2, finalav3]  
 print("the std accuracy(3 tests): " ,(np.std(arr)))  
 vv1=((ta1[0]+ta2[0]+ta3[0])/3)  
 sta1=math.sqrt((((ta1[0]-vv1)\*\*2)+((ta2[0]-vv1)\*\*2)+((ta3[0]-vv1)\*\*2))/3)  
 vv2=((ta1[1] + ta2[1] + ta3[1]) / 3)  
 sta2 = math.sqrt((((ta1[1] - vv2) \*\* 2) + ((ta2[1] - vv2) \*\* 2) + ((ta3[1] - vv2) \*\* 2)) / 3)  
 vv3=((ta1[2] + ta2[2] + ta3[2]) / 3)  
 sta3 = math.sqrt((((ta1[2] - vv3) \*\* 2) + ((ta2[2] - vv3) \*\* 2) + ((ta3[2] - vv3) \*\* 2)) / 3)  
 vv4=((ta1[3] + ta2[3] + ta3[3]) / 3)  
 sta4=math.sqrt((((ta1[3] - vv4) \*\* 2) + ((ta2[3] - vv4) \*\* 2) + ((ta3[3] - vv4) \*\* 2)) / 3)  
 print("the average of actual-yes and predict-yes between the 3 tests is: ",(vv1))  
 print("Standard deviation of actual-yes and predict-yes between the 3 tests is: ",sta1)  
 print("the average of actual-yes and predict-no between the 3 tests is: ", vv2)  
 print("Standard deviation of actual-yes and predict-no between the 3 tests is: ", sta2)  
 print("the average of actual-no and predict-yes between the 3 tests is: ", vv3)  
 print("Standard deviation of actual-no and predict-yes between the 3 tests is: ", sta3)  
 print("the average of actual-no and predict-no between the 3 tests is:: ",vv4)  
 print("Standard deviation of actual-no and predict-no between the 3 tests is: ", sta4)  
 indication=indication+1  
 time\_of\_train\_test=time\_of\_train\_test+1  
  
  
  
end\_time=time.time()  
print("the program run in seconds: ",end\_time-start\_time)  
plt.plot(ave1[2], ave1[0], label="train 1")  
plt.plot(ave2[2], ave2[0], label="train 2")  
plt.plot(ave3[2], ave3[0], label="train 3")  
plt.xlabel('the number of the iterations')  
plt.ylabel('accuracy in %')  
plt.title('the change of accuracy over iterations')  
plt.legend()  
plt.show()  
  
plt.plot(ave1[2], ave1[1], label="train 1")  
plt.plot(ave2[2], ave2[1], label="train 2")  
plt.plot(ave3[2], ave3[1], label="train 3")  
plt.xlabel('the number of the iterations')  
plt.ylabel('Mean Square Error')  
plt.title('the change of Mean Square Error over iterations')  
plt.legend()  
plt.show()