Report on exercise number 1- perceptron

**Cross-validation**

We have divided the data into 3 modules:

1)Training – from the beginning of the file until 66% of the file and the rest of the data utilized for testing.

2)training – from the end of the file we took 66% up to the two thirds of the file, and the rest

Of the data utilized for testing.

3) training – we used the first third of the file from the beginning (33% of the data), and the last third to the end of the file (another 33%) , the rest of the data utilized for testing.

We have normalized our data by taking a column, checking the max number in the column and dividing the rest of the column elements by the max number.

There was a data that was classified as a question mark, in order to overcome this issue by replacing the question mark in integer.

-for example:1.

**Training time**

1. the first module training complete in 0.16 seconds.

2. the second module training completed in 0.15 seconds.

3. the third module training completed in 0.17 seconds.

The purpose of the num\_iteration that was sent as parameter to the “train” function, is to determine the number of times the program will be trained on the input.

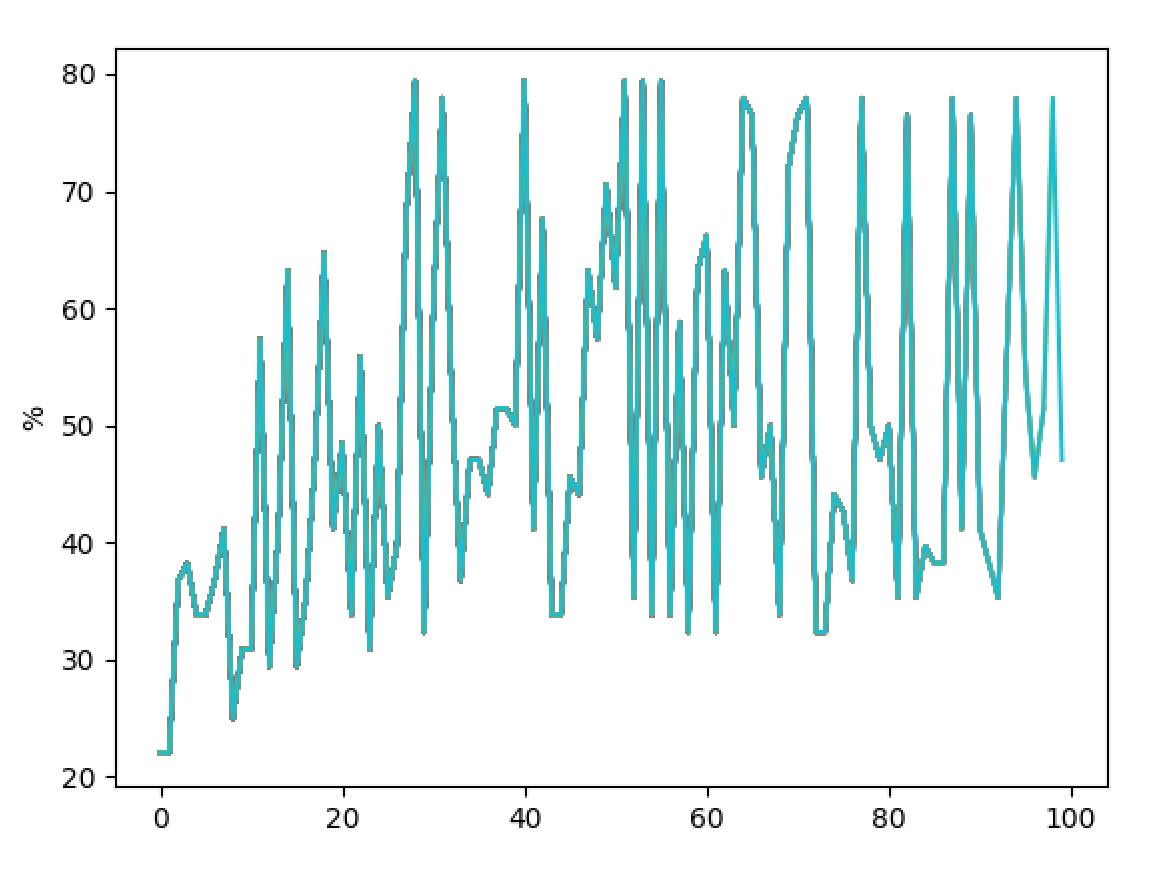
**Performance(for 100 iterations)**

1. 47.05% success in the first module.

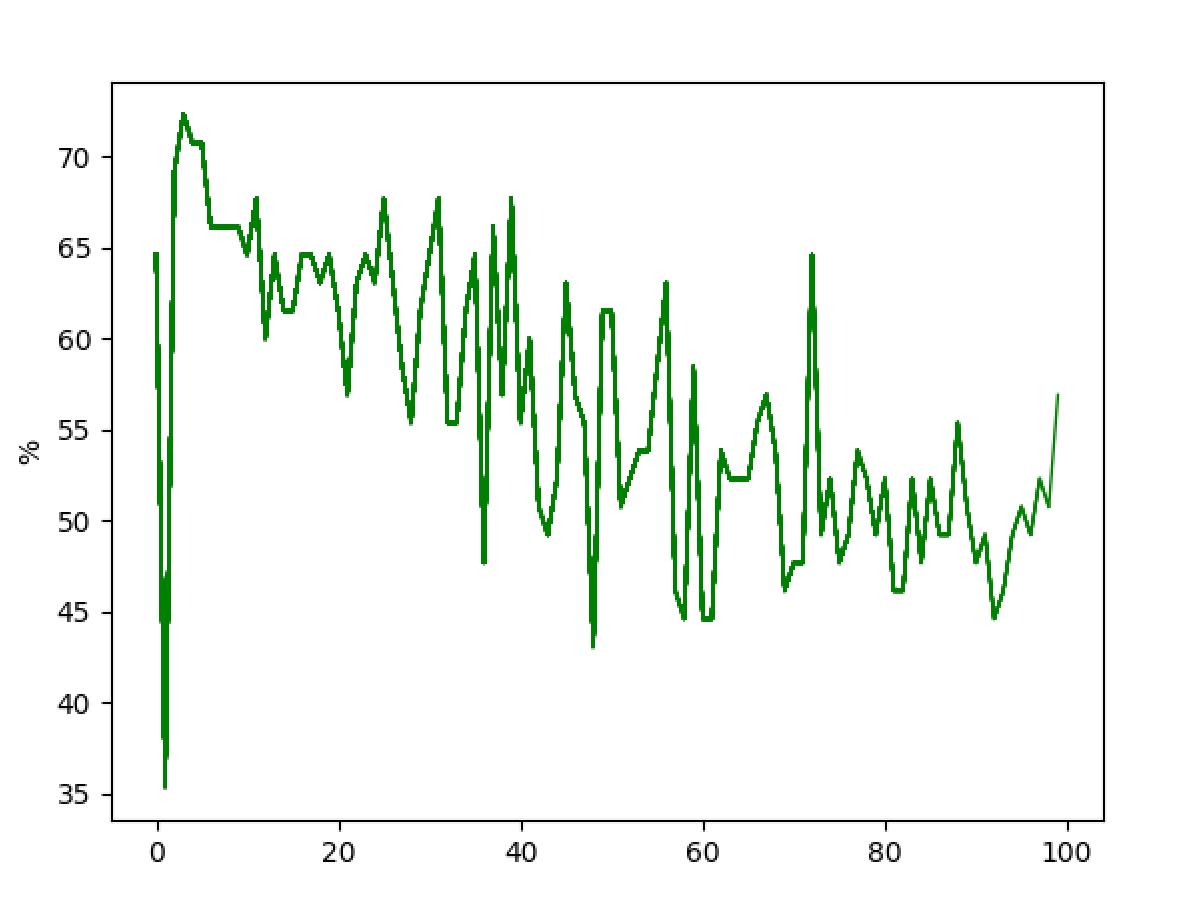
2. 56.92% success in the second module.

3. 81.51% success in the third module.

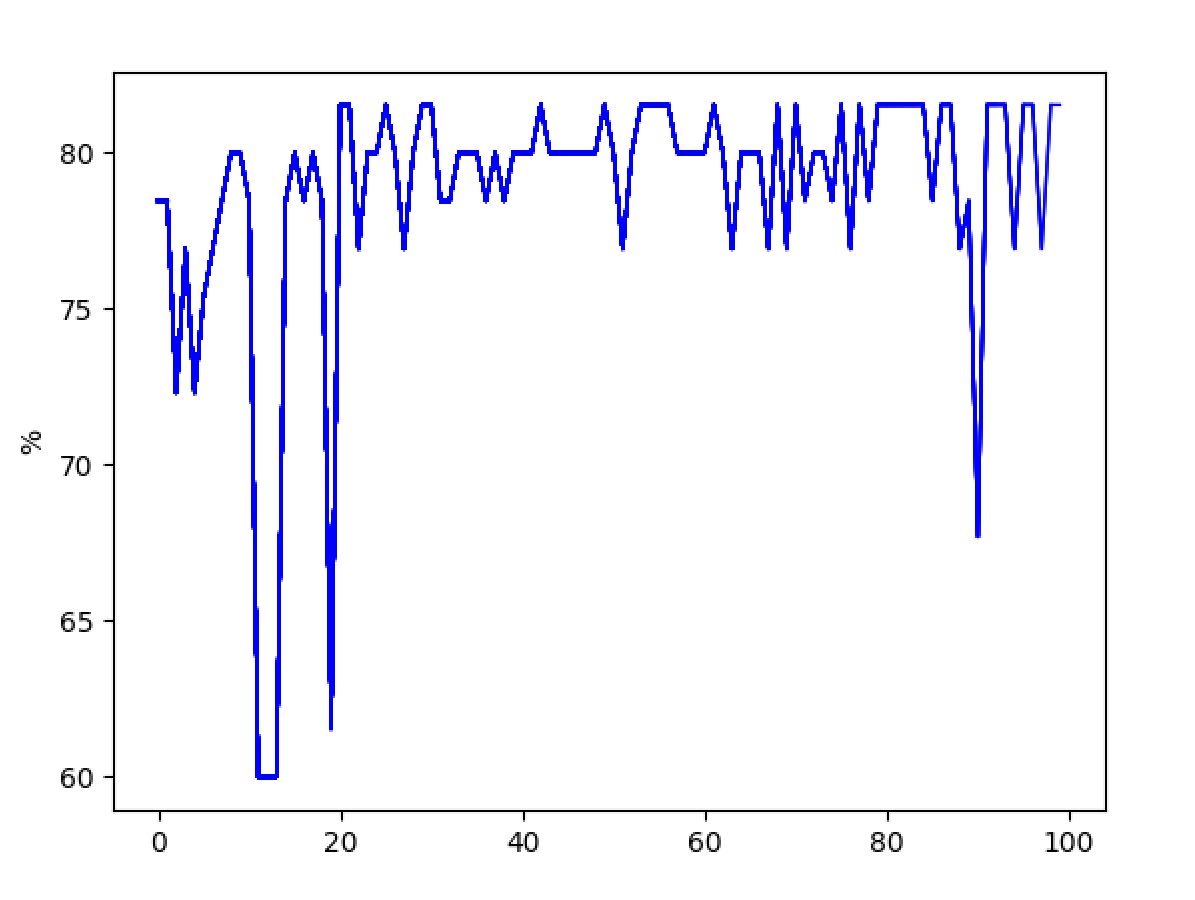
**Test 1:**



**Test 2:**



**Test 3:**



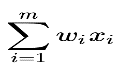
**Summary**:

With the data we got as input on patients with classification R-the disease came back,

Or with classification N-the disease didn’t come back.

There was a need to implement a model where the model will get patients data and base on the previous trained data the model will be able to classify the into 2 classes where there is

A linear separation between those two classes. The model will predict what will happen with the patient based on the new data.

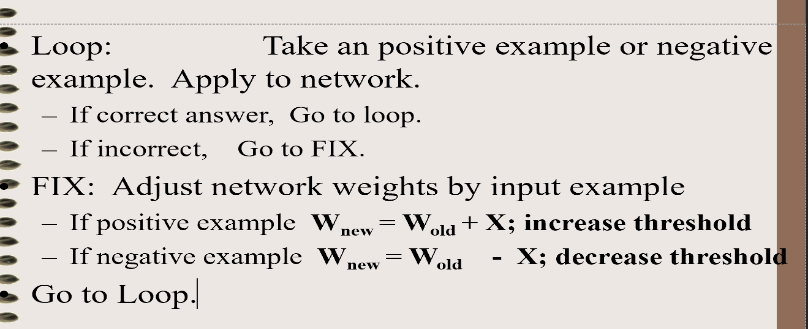
After we have defined weights vector in the set, we initialize it to a zero vector.

We are using the perceptron formula in order to check if the model is making a good or bad prediction on each specific patient he checks.

If the threshold is smaller than zero and the patient was classified as one the disease came back to him, the model will update the existing weight by adding to it the specific input value of the patient from to the patient column.

If the threshold is bigger than zero and the patient was classified as one the disease didn’t come back to him, the model will update the model weights by subtracting the patient input from the patient column.

As we can see from the graphs above, the best data separation was during the third training where we trained the first third of the file from the beginning (33% of the data), and the last third to the end of the file (another 33%) , and the rest of the data we utilized for testing.

the psodo-code mentioned in

the ppt came to out aid in

our code.

we

**Code(python):**

import math  
import numpy as np  
import csv  
import time  
  
  
# X = inputs  
# Y = classification  
  
  
def substract(W, X):  
 tmp = []  
 counter = 0  
 for i in X:  
 tmp.append(W[counter] - X[counter])  
 counter += 1  
 return tmp  
  
  
def add(W, X):  
 tmp = []  
 counter = 0  
 for \_ in X:  
 tmp.append(W[counter] + X[counter])  
 counter += 1  
 return tmp  
  
  
def train(X, Y, num\_iteration):  
 W = np.zeros(len(X[0]))  
 for \_ in range(num\_iteration):  
 for i in range(len(X)):  
 sum = 0  
 for j in range(len(X[i])):  
 sum += W[j] \* X[i][j]  
  
 if Y[i] == 'R' and sum < 0:  
 W = add(W, X[i])   
 elif Y[i] == 'N' and sum >= 0:  
 W = substract(W, X[i])  
  
 return W  
  
  
# X = input test  
# Y = classification  
# W = trained weight  
def test(W, X, Y):  
 result = []  
 for i in range(len(X)):  
 sum = 0  
 for j in range(len(X[i])):  
 sum += float(W[j]) \* float(X[i][j])  
  
 if sum > 0:  
 result.append(1)  
 else:  
 result.append(0)  
 return result  
  
  
def get\_rows(run\_num, num\_rows, is\_for\_training):  
 table = []  
 with open('wpbc.data') as csv\_file:  
 csv\_reader = csv.reader(csv\_file, delimiter=',')  
 row\_counter = 0  
 for row in csv\_reader:  
 row\_counter += 1  
  
 if run\_num == 1:  
 if is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 66:  
 break  
 elif not is\_for\_training:  
 if row\_counter / num\_rows \* 100 < 66:  
 continue  
  
 if run\_num == 2:  
 if not is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 33:  
 break  
 elif row\_counter / num\_rows \* 100 < 33:  
 continue  
  
 if run\_num == 3:  
 if is\_for\_training:  
 if 33 < row\_counter / num\_rows \* 100 < 66:  
 continue  
 if not is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 66:  
 break  
 if row\_counter / num\_rows \* 100 < 33:  
 continue  
  
  
 tmp\_array = []  
 cul\_counter = 0  
 for \_ in row:  
 data = row[int(cul\_counter)]  
 if cul\_counter == 0 or cul\_counter == 1 or cul\_counter == 2:  
 cul\_counter += 1  
 continue  
 data = float(data)  
 tmp\_array.append(data)  
 cul\_counter += 1  
 table.append(tmp\_array)  
 return table  
  
  
def get\_classifications(run\_num, num\_rows, is\_for\_training):  
 arr = []  
 row\_counter = 0  
 with open('wpbc.data') as csv\_file:  
 csv\_reader = csv.reader(csv\_file, delimiter=',')  
 for row in csv\_reader:  
 row\_counter += 1  
  
 if run\_num == 1:  
 if is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 66:  
 break  
 elif not is\_for\_training:  
 if row\_counter / num\_rows \* 100 < 66:  
 continue  
  
 if run\_num == 2:  
 if not is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 33:  
 break  
 elif row\_counter / num\_rows \* 100 < 33:  
 continue  
  
 if run\_num == 3:  
 if is\_for\_training:  
 if 33 < row\_counter / num\_rows \* 100 < 66:  
 continue  
 if not is\_for\_training:  
 if row\_counter / num\_rows \* 100 > 66:  
 break  
 if row\_counter / num\_rows \* 100 < 33:  
 continue  
  
 arr.append(row[1])  
 return arr  
  
  
def get\_num\_rows():  
 counter = 0  
 with open('wpbc.data') as csv\_file:  
 csv\_reader = csv.reader(csv\_file, delimiter=',')  
 for \_ in csv\_reader:  
 counter += 1  
 return counter  
  
  
def get\_biggest\_number(rows, col\_num):  
 biggest\_num = -1  
 for row in rows:  
 current = row[col\_num]  
 if biggest\_num < current:  
 biggest\_num = current  
 return biggest\_num  
  
  
def nirmul(rows, col\_num, number):  
 for row in rows:  
 current = row[col\_num]  
 new\_number = current / number  
 row[col\_num] = new\_number  
 return rows  
  
  
def calculate\_success(answer, classification\_test):  
 success = 0  
 for i in range(len(answer)):  
 current = answer[i]  
 if current == 0:  
 current = 'N'  
 else:  
 current = 'R'  
 if current == classification\_test[i]:  
 success += 1  
 return success  
  
  
def run(run\_num):  
 start\_time = time.time()  
 num\_rows = get\_num\_rows()  
 rows\_train = get\_rows(run\_num, num\_rows, True)  
 classification\_train = get\_classifications(run\_num, num\_rows, True)  
  
 rows\_to\_nirmul = np.array(rows\_train[0])  
  
 for i in range(len(rows\_to\_nirmul)):  
 num = get\_biggest\_number(rows\_train, i)  
 rows\_train = nirmul(rows\_train, i, num)  
  
 weights = train(rows\_train, classification\_train, 100)  
 rows\_test = get\_rows(run\_num, num\_rows, False)  
  
 rows\_test\_to\_nirmul = np.array(rows\_test[0])  
 print("--- Time to execute %s seconds ---" % (time.time() - start\_time))  
  
 for i in range(len(rows\_test\_to\_nirmul)):  
 num = get\_biggest\_number(rows\_test, i)  
 rows\_test = nirmul(rows\_test, i, num)  
  
 classification\_test = get\_classifications(run\_num, num\_rows, False)  
 answer = test(weights, rows\_test, classification\_test)  
 print(answer)  
  
 ans = calculate\_success(answer, classification\_test)  
 print("test", run\_num)  
 print(ans / len(answer) \* 100, "%")  
  
  
run(1)  
run(2)  
run(3)